PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classificati n ^{6} : G06F | A2

(11) International Publication Number: WO 98/24018

(43) Internati nal Publication Date: 4 June 1998 (04.06.98)

(21) International Application Number: PCT/US97/20660

(22) International Filing Date: 13 November 1997 (13.11.97)

(30) Priority Data:

08/752,490	13 November 1996 (13.11.96)	US
08/749,926	13 November 1996 (13.11.96)	US
08/748,645	13 November 1996 (13.11.96)	US

(71) Applicant *(for all designated States except US)*: PUMA TECHNOLOGY INC. [US/US]; 2940 N. First Street, San Jose, CA 95134 (US).

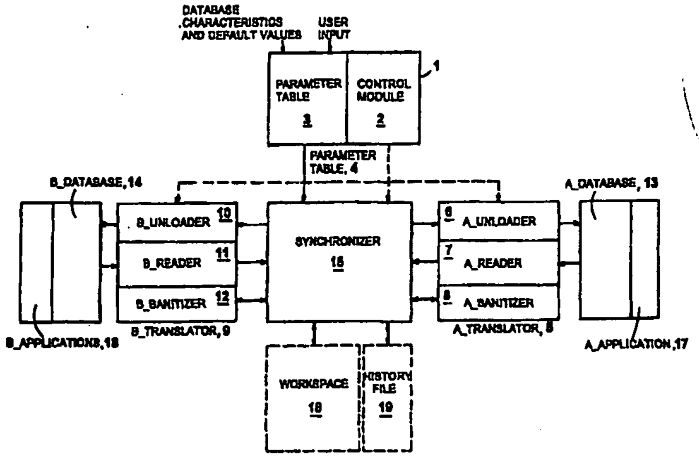
(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: BOOTHBY, David, J. [US/US]; 12 Thoreau Drive, Nashua, NH 03062 (US).

(74) Agent: LEE, G., Roger; Fish & Richardson P.C., 225 Franklin Street, Boston, MA 02110 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: SYNCHRONIZATION OF DATABASES



(57) Abstract

Various methods for synchronizing incompatible databases using a history file containing records representing records of one of the databases at the time of a previous synchronization. A method allows synchronizing databases in which different techniques are used for storing a recurring event. A database in which the recurring event is, for example, stored as a single recurring rec rd can be synchronized with a database in which the same recurring event is stored as a series of individual records. Another method permits comparing records from two different databases where at least one of the databases is subject to rules of data value to which the other database is not subject. The rules of data value are applied to the comparison so that their effect is neutralized and a meaningful comparison can be made. The rules of data value of one database can be used to change copies of the records of the other database. A further method allows synchronizing at least a first and a second database each containing dated records such as events, where the records of the first and second databases are synchronized across a narrow date range narrower than the date range of the records of at least one of the databases. Another method allows synchronizing two or more databases with a single database. In that case, for example, synchronized rec rds are tagged with database identifying codes which indicated the database from which the records originated.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

SYNCHRONIZATION OF DATABASES

Background

This invention relates to synchronizing incompatible databases.

Databases are collections of data entries which are organized, stored, and manipulated in a manner specified by applications known as database managers (hereinafter also referred to as "Applications"). The manner in which database entries are organized in a database is known as the data structure. There are generally two types of database managers. First are general purpose database managers in which the user determines (usually at the outset, but subject to future revisions) what the data structure is. These Applications often have their own programming language and provide great flexibility to the user. Second are special purpose database managers that are specifically designed to create and manage a database having a preset data structure. Examples of these special purpose database managers are various scheduling, diary, and contact manager Applications for desktop and handheld computers. Database managers organize the information in a database into records, with each record made up of fields. Fields and records of a database may have many different characteristics depending on the database manager's purpose and utility.

Databases can be said to be incompatible with one another when the data structure of one is not the same as the data structure of another, even though some of the cont nt f the records is substantially the sam . F r example, n database may store names and addresses in the f llowing fi lds: FIRST_NAME, LAST_NAME, and

ADDRESS. Anoth r database may, however, store the sam information with the following structure: NAME, STREET_NO., STREET_NAME, CITY_STATE, and ZIP. Although the content of the records is intended to contain the same kind of information, the organization of that information is completely different.

It is often the case that users of incompatible databases want to be able to synchronize the databases. For example, in the context of scheduling and contact manager Applications, a person might use one Application on the desktop computer at work while another on his handheld computer or his laptop computer at home. It is desirable for many of these users to be able to synchronize the entries on one with entries on another. However, the incompatibility of the two databases creates many problems that need to be solved for successful synchronization. The U.S. patent and copending patent application of the assignee hereof, IntelliLink Corp., of Nashua, New Hampshire (U.S. Patent No. 5,392,390; U.S. Application, Serial No. 08/371,194, filed on January 11, 1995, incorporated by reference herein) show two methods for synchronizing incompatible databases and solving some of the problems arising from incompatibility of databases. However, other problems remain.

One kind of incompatibility is when one database manager uses recurring records. Recurring records are single records which contain information which indicates that the records actually represent multiple records sharing some common information. Many scheduling Applications, for example, permit as a single record an event which occurs regularly over a period of time. Instances of such entries are biweekly committee meetings or we kly staff lunches. Other scheduling Applications do not us th s typ s of records. A user has to cr ate equivalent ntries by cr ating a separat record for each instance f these recurring events.

Various problems arise when synchronizing these types of records. Let us consider a situation when Application A uses recurring records while Application B does not. A synchronizing application must be able to create multiple entries for B for each recurring entry in A. It also must be able to identify some of the records in database B as instances of recurring records in database A. Also, many Applications which allow recurring records also permit revision and editing of single instances of recurring records without affecting the master recurring record. Moreover, single instances of a recurring event in Application B may be changed or deleted. The recurring master may also be changed which has the effect of changing all instances. These changes make it harder to identify multiple entries in database B as instances of a recurring record in database A. Moreover, synchronization must take these changes into account when updating records in one or the other database.

Another kind of incompatibility arises in the case of database managers which impose restrictions and rules on the content of records. For example, the length of text entered by a user into a field may be limited (e.g. to save storage space) or the values permitted may be limited (e.g. to impose psychological discipline, as in limiting priority values in To Do lists to 3). The Application may also require that all text be UPPERCASE. Other limitations may be more complicated, in the form of complex rules and requirements. In Microsoft® Schedule+, for example, Tasks records have four fields called StartDate, EndDate, AlarmDate, and AlarmFlag. The contents f these fi lds must follow a set of rules. If StartDate and EndDate are both blank, AlarmDate must be blank and AlarmFlag must be set to FALSE, because Schedule+ do s n t allow alarms for undated Tasks. If StartDate is not blank, the EndDate should not be blank either. Because of these rules, issues arise with respect to how records from these incompatible databases compare.

Another kind of incompatibility arises in the case of databases which run on computer systems with very limited storage capacity, such as handheld computers. It is often desirable to synchronize the databases on these devices with databases on larger computers such as desktop computers which have much higher storage capacity. However, a straight synchronization between the Applications on the two devices may result in storage capacity of the smaller devices being mostly consumed with the records from the larger device, rendering the smaller device inoperable.

Summary

In one general aspect, the invention provides a technique for synchronizing databases in which different techniques are used for storing a recurring event. A database in which the recurring event is, for example, stored as a single recurring record can be synchronized with a database in which the same recurring event is stored as a series of individual records. The individual records are processed to form a synthetic recurring record representing the set of individual records, and synchronization decisions are based on a comparison of the synthetic record to the recurring record of the other database. Following synchronization, the synthetic rec rd can be "fanned" back into the individual records to update the database containing individual records, and the updated r curring record can b written back to the oth r databas . In this way, the invention av ids the problems ncount red with prior methods, in which synchronization resulted in a recurring record being transformed into a series of individual records.

In another general aspect, the invention features a computer implemented method of synchronizing at least a first and a second database, wherein the manner of storing a set of recurring instances differs between the first and second databases, and at least the first database uses a recurring record to store the set of recurring instances. A plurality of instances in the second database are processed to generate a synthetic recurring record representing recurring instances in the second database, the synthetic recurring record of the second database is compared to a recurring record of the first database, and synchronization is completed based on the outcome of the comparison.

Preferred embodiments of these aspects of the invention may include one or more of the following features: Completing synchronization may include adding, modifying, or deleting the synthetic recurring record or the recurring record. Following synchronization, the synthetic recurring record may be fanned back into a plurality of single instances. The set of recurring instances may be stored in the second database as a plurality of single instances. The set of recurring instances may be stored in the second database as a recurring record having a different record structure than the recurring record of the first database. A history file may be stored containing a record representative of the presence of a recurring record or a synthetic recurring record in past synchronizations.

In y t another general aspect, the invention allows comparison of r cords from two different databases where at least one of the databases is subj ct to rules of data value to which the other database is not subject. The rules of data value of ne database are used to change copies of the records of the other database so that a meaningful comparison can be made.

The invention features a computer implemented method of synchronizing records of first and second databases, wherein at least one field of records of the first database is subject to a first rule of data value to which the corresponding field of records of the second database is not subject. The first rule of data value of a field of the first database is used to modify copies of the content of corresponding fields of records of the second database. Thereafter, the content of the modified copies is compared to the content of the corresponding field of the first database, and synchronization actions are taken based on the outcome of the comparison.

In preferred embodiments of this aspect of the invention, at least one field of records of the second database is subject to a second rule of data value to which the corresponding field of records of the second database is not subject, and the second rule of data value is used to modify copies of the content of corresponding fields of records of the first database; and the content of modified copies of the content of the first database is compared to modified copies of the content of the second database.

In another general aspect, the invention may take into account rules of data value at the time of comparison. For example, two text fields may be compared only up to the character limit of one of them.

In one other general aspect, the invention provides a method of synchronizing multiple databases of different Applications. A database's record, when written in another database may be tagged with a uniqu mark identifying the sourc of th record. Th se tags may be us d to filt r out only those r cords which should be synchronized. Th tags may be attached when th records are unloaded to the databases.

In yet another general aspect, the invention solves the difficulty of synchronizing databases in which events are maintained across different date ranges. A date range is set for which synchronization will take place. Records falling outside of the date range are not synchronized. The date range of the prior synchronization is stored, and a current synchronization is performed across the combination of the current and prior date ranges. The problems of synchronization software attempting to fill a smaller capacity device with events across a wide date range that can only practically be stored on a larger capacity device are avoided.

In this aspect, the invention features a computer implemented method of synchronizing at least a first and a second database each containing dated records such as events, wherein the records of the first database extend across a narrow date range narrower than the date range of the records of the second database. A prior synchronization is performed across a prior date range set using the date of the prior synchronization and the narrow date range. The date range of the prior synchronization is stored, along with the history file containing information representative of the content of the databases following the prior synchronization. When a current synchronization is performed, it is performed across a date range that combines the prior date range with a current date range set using the date of the current synchronization and the narrow date range.

Th invention may be implemented in hardware r software, or a combination of both. Preferably, the technique is implemented in computer programs executing on programmable computers that ach include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. Program code is applied to data entered using the input device to perform the functions described above and to generate output information. The output information is applied to one or more output devices.

Each program is preferably implemented in a high level procedural or object oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

Each such computer program is preferably stored on a storage medium or device (e.g., ROM or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described in this document. The system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

Other features and advantages of the invention will become apparent from the following description of preferred embodiments, including the drawings, and from the claims.

<u>Brief Description of the Drawing</u>

Figure 1 is a schematic drawing of the various modules constituting th preferred embodiment.

Figur 2 is a representation of th Workspace data array.

Figure 3 is the pseudocode for th Translation Engine Control Module.

Figure 4 is the pseudocode for generating the parameter Table.

Figure 5 is the pseudocode for fanning a recurring record.

Figure 6 is the pseudocode for the Synchronizer loading the History File.

Figure 7 is the pseudocode for matching key fields (Key_Field_Match).

Figure 8 is the pseudocode for loading records of B_Database into Workspace.

Figure 9 is the pseudocode for A_Sanitization of B_Database records in Workspace.

Figure 10 is the Pseudocode for a specific example of a rule of data value used for sanitization.

Figure 11 is the pseudocode for orientation analysis.

Figure 12 is the pseudocode for Conflict Analysis And Resolution (CAAR).

Figure 13 is the pseudocode for analyzing unique ID bearing Fanned Instance Groups (FIGs).

Figure 14 is the pseudocode for expanding CIGs created from unique ID bearing records.

Figure 15 is the pseudocode for finding weak matches for a record.

Figure 16 is the pseudocode for finding matches between recurring items and non_unique ID bearing instances.

Figure 17 is the pseudocode for completing Same Key Group (SKG) analysis.

Figure 18 is the pseudocode for setting the Maximum_CIG_Size for ev ry CIG analyz d in Figure 17.

Figur 19 is the pseud code for setting CIG_Types.

Description

Fig. 1 shows the relationship between the various modules of the preferred embodiment. Translation Engine 1 comprises Control Module 2 and Parameters Table Generator 3. Control Module 2 is responsible for controlling the synchronizing process by instructing various modules to perform specific tasks on the records of the two databases being synchronized. The steps taken by this module are demonstrated in Fig. 3. The Parameters Table Generator 3 is responsible for creating a Parameter_Table 4 which is used by all other modules f r synchr nizing th databases. D tails of the Parameter_Table are describ d in more d tail below. The Synchronizer 15 has primary responsibility for carrying out the core synchronizing functions. It is a table-driven cod which is capabl of synchronizing various types of databases whose characteristics are provided in the Parameter_Table 4. The Synchronizer creates and uses the Workspace 16, which is a temporary data array used during the synchronization process.

A Translator 5 (A_Translator) is assigned to the A_database 13 and another Translator 9 (B_Translator) to the B_database 14. Each of the database Translators 5 and 9 comprises three modules: Reader modules 6 and 10 (A_Reader and B_Reader), which read the data from the databases 13 and 14; Unloader modules 8 and 12 (A_Unloader and B_Unloader), which analyze and unload records from the Workspace into the databases 13 and 14; and Sanitizing modules 7 and 11 (A_Sanitizer and B_Sanitizer), which analyze the records of the other database loaded into the Workspace and modify them according to rules of data value of its own database. In the preferred embodiment, the modules of the A_Translator 5 are designed specifically for interacting with the A_database 13 and the A_Application 17. Their design is specifically based on the record and field structures and the rules of data value imposed on them by the A_Application, the Application Program Interface (API) requirements and limitations of the A_Application and other characteristics of A_Database and A_Application. The same is true of the modules of B_Translator 9. These Translators are not able to interact with any other databases or Applications. They are only aware of the characteristics of the database and the Application for which they have been designed. Therefore, in the preferred embodiment, when the user ch oses two Applications for synchronizati n, the Translation Engine chooses the two Translat rs which are able t interact with those Applications. In an alt rnat emb diment, the translator can be designed as a table-driven code, where a general Translator is able to interact with a variety of Applications and databases based on the parameters supplied by the Translation Engine 1.

Referring to Figs. 1, 2 and 3, the synchronization process is as follows. The Parameter_Table 4 is generated by the Parameter Table Generator 3. The Synchronizer 15 then creates the Workspace 16 data array and loads the History File 19 into the Workspace 16. The B_Reader module 11 of the B_Translator reads the B_database records and sends them to the Synchronizer for writing into the Workspace. Following the loading of B_Database records, the A_Sanitizer module 8 of the A_Translator 5 sanitizes the B_Records in the Workspace. The A_Reader module 7 of the A_Translator 5 then reads the A_Database records and sends them to the Synchronizer 16 for writing into the Workspace. The B_Sanitizer module 12 of the B_Translator 9 then sanitizes the A_Records in the Workspace. The Synchronizer then performs the Conflict Analysis and Resolution (CAAR) on the records in Workspace. At the end of this analysis the user is asked whether he/she would like to proceed with updating the A_ and B_databases. If so, the B_Unloader module of the B_Translator unloads the appropriate records into the B_database. The A_Unloader module 6 then performs the same task for the A_Database. Finally, the Synchronizer creates a new History File 19.

Fig. 3 is the pseudocode for the preferred embodiment of the Control Module 2 of the Translation Engine 1. Control Module 2 first instructs the Parameter Table Generator 3 of the Translation Engine 1 to create th Param ter_Tabl (Step 100). Fig. 4 is th pseud code for the pr ferr d embodiment of the Paramet r Tabl G n rat r m dule 3. The user is first asked to ch se wh ther t us a previously chosen and stored set f preferences or to nter a n w s t of preferences (St p 150). Steps 151-165 show the steps in which the user inputs his/her new preferences. In step 152, the user chooses whether to perform a synchronization from scratch or an incremental synchronization. In a synchronization from scratch, synchronization is performed as if this was the first time the two databases were being synchronized. In an incremental synchronization, the History File from the previous file is used to assist with synchronization. The user will likely choose incremental synchronization if there has been a prior synchronization, but the user may choose to synchronize from scratch where the user would like to start with a clean slate (perhaps due to significant change in the nature of the data in the databases). The user then selects the two Applications and related databases (A_Database and B_Database) to be synchronized (step 153). The user then chooses (step 154) whether the Synchronizer should use the default field mapping for those two databases during synchronization or the user will modify the field mapping. Field mapping is generally described in U.S. Patent No. 5,392,390 (incorporated by reference). In accordance with the user's preferences, the Parameter Table Generator then stores the appropriate A_Database to B_Database fields map (A→B_Map) and B_Database to A_Database fields map (B→A_Map) in the Parameter_Table (Steps 155-158 and 159-163, accordingly).

If in step 150 the user selected to use previously chosen and stored set of preferences (steps 166-171), those preferences are loaded and stored in the Parameter_Table (steps 169-170).

In case of date b aring records such as appointments and ToDo lists, the user enters th date range for which the user wants the records to be synchronized (step 172). The preferred embodiment allows th user to use relative date ranges (Automatic_Date_Range) (substeps 171 (a) and (b)). For example, the user can select the date range to be 30 days into the past from today's date and 60 days into the future from today's date. The Parameter Table Generator 3 then calculates and stores in the Parameter_Table the Start_Current_Date_Range and End_Current_Date_Range values, the two variables indicating the starting point and the ending point of the date range for the current synchronization session (step 173-174).

In steps 174 and 175, various parameters identifying the characteristics of the A_Database and Application and B_Database and Application are loaded from a database (not shown) holding such data for different Applications. These are in turn stored in the Parameter_Table. One of the sets of parameters loaded and stored in the Parameter_Table is the Field_List for the two databases. The Field_List_A and Field_List_B contain the following information about each field in the data structure of the two databases:

1. Field name.
2. Field Type.
3. Field Limitations.
4. No_Reconcile Flag.
6. Key_Field Flag.
7. Mapped_Field Flag.

Field name is the name given to the field which the Translator for this Application uses. This name may also be the name used by the Application. Field Type identifies to the Synchronizer 15 the nature of the data in a field, e.g., Data, Time, Boolean, Text, Number, or Binary. The Field Name does not supply this informati n to the Synchroniz r. Field Limitati ns identifies the various limitations th database manager imp ses on the cont nts of a field. Th se limitations include: maximum length of text fields, whether th text field must b in upper-case, range of permissible values (for example, in ToDo records priority field, the range of permissible values may be limited from 1 to 4), and whether a single line or multiple line field.

No_Reconcile flag indicates whether a field is a No_Reconcile field, meaning that it will not be used to match records nor will it be synchronized although it will be mapped and possibly used in synchronization. Almost all fields will not be designated as No_Reconcile. However, sometimes it is necessary to do so. Key_Field flag indicates that a field should be considered as a key field by the Synchronizer 15.

Key fields are used by the Synchronizer in various stages of synchronization as will be discussed in detail below. The decision of identifying certain fields as key is based on examining the various Applications to be synchronized, their data structure, and the purpose for which the database is used. Such examination reveals which fields would best function as key fields for synchronization. For example, for an address book database, the lastname, firstname, and company name field may be chosen as key fields. For Appointments, the date field and the description field may be chosen as key fields.

Mapped_Field flag indicates whether a field is mapped at all. The Synchronizer uses this flag to determine whether it should use the A→B_Map or B→A_Map to map this field. Unlike a No_Reconcile field, an unmapped field will not be carried along through the synchronization.

Another set of param ters in the Param ter_Table identify the Translator Modules 13, 14 for the two Applicati ns which the user has selected. Because each

Application is assigned its own Translator, it is necessary to identify to the Command Module and the Synchronizer which Translators should be used.

In step 102 of Fig. 1, the Translation Engine instructs the Synchronizer to load the History File. History File is the file which was saved at the end of last synchronization. It contains the history of the previous synchronization which is necessary for use with the current synchronization in case of Incremental Synchronization. Records from the A_Database and B_Database are analyzed against the records of the history file to determine the changes, additions, and deletions in each of two databases since last synchronization and whether additions, deletions, or updates need to be done to the records of the databases. Referring to Fig. 5, in steps 200-201, the Synchronizer finds the appropriate History file to be loaded. If Synchronization_from_Scratch flag is set, the History File is deleted (step 203). If no History File is found, the synchronization will proceed as if it was a synchronization from scratch (step 204). If the Field Lists stored in the History File are not the same as the current Field Lists in the Parameter_Table, or the mapping information is not the same, the synchronization will proceed as synchronization from scratch because the differences indicate that the History File records will not properly match the database records (steps 206-209).

In step 210, the Synchronizer uses the Field_List for database B to create the Workspace 16. It is a large record array which the Synchronizer uses during synchronization. Referring to Fig. 2, Workspace 16 consist of two sections. First, the Synchronizer uses the Fi ld List for th B_Database to mak a record array 21 which has all the characteristics of the B_Databas r cord structur . In addition, in each record in the

Workspac , certain internal fields ar added. On field is _subtype containing Origin Tags. Two other fi lds, called Rep_Basic and Rep_Excl, are included for all Appointment and ToDo Sections. The Rep_Basic field gives a full description of the recurrence pattern of a recurring record. It includes the following parameters:

1. Basic_Repeat_Type
2. Frequency
3. StopDate
4. other parameters
5. Rep_Excl

Basic_Repeat_Type contains the variable which indicates whether the recurring record is a daily, weekly, monthly (same date each month), monthly by position (e.g., 3rd Friday of each month), yearly (e.g., July 4th each year), yearly by Position (e.g., 3rd Friday of September each year), quarterly, etc. This variable is set to No_Repeat for non-recurring records.

Frequency indicates whether the pattern is, for example, for every week, every other week, etc. StartDate and StopDate show the first date and last date in the pattern. Some other parameters in the Rep_Basic include, for example, a list of days to be included for the pattern (e.g. I plan to hold a weekly staff meeting every Thursday starting November 15, 1997.)

Rep_Excl is the exclusion list. It is a list of dates which at some point belonged to the recurring record, but have since been deleted or modified and no longer are an event represented by the recurring record.

Since some databases do not provide for recurring types of records, the synchronization process sometimes must create single records for each of the instances of a recurring record for those databases. For example, for a recurring lunch every Thursday, the synchronization must produce a single record for ach Thursday in such a database. This is acc mplish d by the pr cess f fanning which us s R p_Basic. Each of those instances is called a fanned instance. Fig. 6 sets out the preferred embodiment of the process of fanning a record.

Fanning of recurring records also takes into account another set of considerations regarding date range limitations and usefulness of instances to the user.

First, fanning is limited to the applicable date range. Second, the number of fanned instances is limited. When synchronizing Databases A and B, the preferred embodiment permits different sets of limits on fanned instances to be established for each Database. This, for example, assists with managing storage capacity of a memory-constrained handheld device when being synchronized with a database on a desktop PC.

If the current Date Range is large enough to accommodate more than the maximum number of instances which might be generated, those instances will be chosen which are likely to be most useful to the user. In the preferred embodiment, it is assumed that future instances are always more useful than past instances, that near future instances are more useful than distant future instances, and that recent past instances are more useful than distant past instances. Therefore, based on these assumptions, a fanning date range is calculated (Fig. 6, step 236).

Referring to Fig. 2, in the second step of creating the Workspace, the Synchronizer establishes an Extended Index Array 20 which has an index entry associated with each entry in the record array. Each index contains the following variables:

1. Next_In_CIG:
2. Next_In_SKG:
3. Next_In_FIG

4. K y_Field_Hash
5. A_Unique_ID_Hash
6. B_Unique_ID_Hash
7. Non_Key_Field_Hash
8. Non_Date_Hash
9. Exclusion_List_Hash
10. Start_Date&Time
11. End_Date&Time
12. Various bit flags

Next_In_CIG is a linkage word, pointing to next member of the same Corresponding Item Group (CIG). A CIG is a group of records, one from each database and the History File, if applicable, which represent the same entry in each of the databases and the History File. There may be one, two or three records in a CIG. Next_In_SKG is a linkage word, pointing to next member of the Same Key Fields Group (SKG). An SKG is a group of records having the same key fields. Next_In_FIG is a linkage word, pointing to the next member of the Fanned Instances Group (FIG). A FIG is the group of fanned instances which correspond to a single recurring record.

Key_Field_Hash is hash of all Key_Fields. A_unique_ID_Hash is hash of unique ID, if any, assigned by A_Database. B_unique_ID_Hash is hash of unique ID, if any, assigned by B_Database. Non_Key_Field_Hash is hash of all Non-Key Match Field, a Match Field being any mapped field which is not flagged as No_Reconcile. Non_Date_Hash is hash of all Non-Date Non-Key Match Fields. Exclusion_List_Hash is hash of recurring record's exclusion list.

Start_Date&Time and End_Date&Time are used for Appointment and ToDo type record only, indicating the start and end dat and time of th record. They are used t speed up comparing functions throughout the synchr nization.

Hash values are also us d to speed up the process of comparis n. The preferred embodiment uses integer hashes. Hash value computation takes into account certain rules of data value for fields, as will be described in more detail below.

In the preferred embodiment, the record array 21 is stored on magnetic disk of a computer whereas the Extended Index 20 is held resident in memory. The Extended Indexes have record pointer fields which point to each of the records on the disk file.

The Control Module 2 now instructs the synchronizer to load the History File into the Workspace (Fig. 3, step 102). Referring to Fig. 6, the synchronizer loads the records beginning in first available spot in the Workspace (step 211). The Synchronizer then performs an analysis on each of the records and resets some of the values in the records (steps 212-228). In case of recurring records, if any of the instances is within the current date range, then the recurring record itself will be considered within the current date range (steps 217-227). The synchronizer then builds SKGs by finding for each history record one record which has matching key fields and by placing that record in the SKG of the history record (step 215-216). Referring to Fig. 7, steps 250-258 describe the Key_Field_Match function used for matching records for SKG.

When comparing two records or two fields, in the preferred embodiment, the COMPARE function is used. The COMPARE function is intelligent comparison logic, which takes into account some of the differences between the rules of data value imposed by the A_Application and the B_Applicati n n th ir r sp ctiv databases. S me examples ar as follows. The COMPARE function is insensitive t upper and lower case letters if case insensitive field attribute is present. Because some Applications require entries to be in all capital l tt r, the COMPARE function ignores the differences between upper and lowercase letters. The COMPARE function takes into account any text length limitations. For example, when comparing "App" in the A_Database and "Apple" in the B_Database, the COMPARE function takes into account that this field is limited to only 3 characters in the A_Database. It also takes into account limits on numerical value. For example, priority fields in the A_Application may be limited to only values up to 3, whereas in the B_Application there may not be any limitation. The COMPARE function would treat all values in B_records above 3 as 3.

The COMPARE function may ignore various codes such as end of line characters. It may strip punctuation from some fields such as telephone numbers and trailing white space from text fields (i.e "Hello   " is treated as "Hello"). It also considers field mapping. For example, if the only line that is mapped by the A→B_Map is the first line of a field, then only that line is compared. When comparing appointment fields, because different databases handle alarm date and time differently when Alarmflag is false, the COMPARE function treats them as equal even though the values in them are not the same. It skips Alarm Date and Time, if the Alarm Flag is False. It also ignores exclusion lists when comparing recurring records.

In an alternate embodiment, the COMPARE function may take into account more complicated rules for data value of the two Applications, such as the rules for data value imposed by Microsoft Schedule+, described above. Such a COMPARE function may be implemented as a table driven code, the table containing the rules imposed by the A_Application and the B_Application. Because th

COMPARE function has a sp cific comparis n logic and takes into account a number of rules, the hashing logic must also follow the same rules. It should be noted that the COMPARE function is used throughout the preferred embodiment for field comparisons.

Now that the History File is loaded into the Workspace, the Control Module 2 instructs the B_Translator 13 to load the B_Database records (Fig. 3, step 103). Referring to Fig. 8, steps 300-308, the B_Reader module 11 of the B_Translator 13 loads each B_record which has the right Origin Tag, which will be explained in more detail below.

The record must also be within the loading date range, which is a concatenation of the previous and current date ranges. The B_Translator sends these records to the Synchronizer which in turn stores them in the Workspace. When synchronizing with a date range limitation, all records which fall within either the previous or the current date ranges are loaded. The current date range is used during unloading to limit the unloading of the records to only those records which fall within the database's current date range. In an alternate embodiment of the invention, each database or Application can have its own date range for each synchronization.

Most Applications or databases permit record-specific and field-specific updates to a Database. But some Applications or databases do not. Instead the Translator for these Application must re-create the whole database from scratch when unloading at the end of synchronization. These databases are identified as Rebuild_All databases. To accommodate this requirement all rec rds fr m such a database must be load d int the Workspac , so that th y can later be used to rebuild th whole databas . Thes databases rec rds, which w uld otherwise have b en filter d ut by the date rang or the wrong origin tag filters, are instead marked with special flag bits as Out_Of_Range or Wrong_Section_Subtype. These records will be ignored during the synchronization process but will be written back unmodified into the database from which they came by the responsible Unloader module 6, 10.

Control Module 2 next instructs the A_Translator 5 to sanitize the B-records. Referring to Fig. 9, steps 350-361, the A_Sanitizer module 8 of the A_Translator 5 is designed to take a record having the form of an A_Record and make it conform to the specific rules of data value imposed by the A_Application on records of the A_Database. A_Sanitizer is not aware which database's field and records it is making to conform to its own Application's format. It is only aware of the A_Application's field and record structure or data structure. Therefore, when it requests a field from the sanitizer using the A_Database field name, it is asking for fields having the A_Database data structure. The Synchronizer, in steps 375-387, therefore maps each record according to the B→A_Map. In turn, when the Synchronizer receives the fields from the A_SANITIZER, it waits until it assembles a whole record (by keeping the values in a cache) and then maps the record back into the B format using the A→B_Map.

How a record or a field is sanitized in step 354 and 357 depends on the rules of data value imposed by the A_Application. For example, all of the logic of intelligent comparison in the COMPARE function described above can be implemented by sanitization. However, sanitization is best suited for more complex or unique types of database rules for data value. For xampl , consider th Sch dule+ rules r garding alarm bearing Tasks r cords describ d above. Fig. 10 shows a sanitizati n m thod for making records of incompatibl databas s c nf rm to the requirements of Schedule+. Without sanitization, when a Tasks record of a Schedule+ database is compared to its corresponding record in another database, the Tasks record may be updated in fields which should be blank according to the Schedule+ rules of data value. Such an update may possibly affect the proper operation of Schedule+ after synchronization.

Referring to Fig. 11, following sanitization of all B_Records into the Workspace, the Synchronizer sets the values for the Extended Index of each record based on the record's values (steps 451-459). Also if the records in the B_Database bear a unique ID, and matches for those unique IDs are found in the H_Records in the Workspace, the two records are joined in a CIG because they represent the same record in both History File and B_Database (step 462). The record is also joined to an SKG it may belong to (step 464). The loading of B_Records is now complete.

The Control Module 2 of the Translation Engine 3 now instructs the A_Translator 5 to load the records from the A_Database (step 105). The loading process for the A_Records is the same as the loading process for the B_Database, except for some differences arising from the fact that records in the Workspace are stored according to the B_Database data structure. Therefore, as the synchronizer 15 receives each A_record from the A_Reader module 7 of the A_Translator 5, the Synchronizer maps that record using the A→B_Map before writing the record into the next available spot in the Workspace. Since the A_records are mapped into the B_Record format, when the B_Sanitizer is instructed by the Control Module 2 to begin sanitizing th s r cords and starts asking f r them from the synchr niz r, th y already hav the B_Database format. Ther fore, th synchronizer 15 does not n ed t map them bef r s nding them to the B_Sanitizer module 12 of th B_Translator 19. For the same r ason, th re is no need for them to be mapped once they are sent back by the B_Sanitizer after having been sanitized. Once all the records are loaded, the records will undergo the same orientation analysis that the B_Records underwent (Fig. 11).

At this point, all records are loaded into the Workspace. SKGs are complete since every record at the time of loading is connected to the appropriate SKG. CIGs now contain all records that could be matched based on unique IDs. At this point, the records in the Workspace will be analyzed according to Conflict Analysis and Resolution ("CAAR") which is set out in Fig. 12 and in more detail in Figs. 13-18 and corresponding detailed description.

First, in step 500, ID bearing fanned instances in the History File records are matched to the fanned instances in the ID bearing database from which they came. The records from the database which have remained unchanged are formed into a new FIG. A new Synthetic Master is created based on those records and joined to them. The records which have been changed or deleted since last synchronization are set free as single records. They also result in a new exclusion list being created based on an old exclusion list and these new single records.

Second, in step 501, matches are sought for the ID based CIGs which are the only CIGs so far created in order to increase the membership of those CIGs. Preferably an exact all fields match is sought between current members of a CIG and a new one. Failing that, a weaker match is sought.

Third, in step 502, master/instanc s match is sought between r curring r cords and non-unique ID bearing instances by trying to find the larg st group of instanc s which match certain valu s in the Recurring Master.

Fourth, in step 503, the items remaining in the SKGs are matched up based on either exact all field match or master/instance match, or a weaker match.

Fifth, in step 501, the appropriate CIG_Types are set for all the CIGs. CIG_Types will determine what the outcome of unloading the records will be.

Referring to Fig. 13, first step in CAAR is analyzing unique ID bearing Fanned Instance Groups. This analysis attempts to optimize using unique IDs assigned by databases in analyzing fanned instances of recurring records.

The analysis is performed for all Recurring Masters (i.e. all recurring records) which have ID-bearing fanned instances (or FIG records) in the H_File (step 550). All FIG records in the History File associated with a Recurring Master are analyzed (steps 551-559). They are all removed from the SKG. If a FIG record is a singleton CIG, it means that it was deleted from the database since the previous synchronization. Therefore, it is added to the New_Exclusion_List (step 553). If a FIG record is a doubleton and is an exact match, it means that the record was not modified since the previous synchronization. In this case, the record from the database is also removed from SKG (step 555). If a FIG record is a doubleton but is not an exact match for its counterpart in the database, it means that the record was changed in the database. The History File record is treated as a deletion and therefore added to the New_Exclusion_List. The modified record in the database, which does n t match th r curring rec rd any longer, is tr at d as a free standing record un-associated with th Recurring Mast r (step 557).

Upon analysis of all FIG records, a new rec rd, the Synthetic Master, is creat d and join d in a CIG with the Recurring Master (step 231-236). The Synthetic Master has the same characteristics as the Recurring Master, except that it has a new exclusion list which is a merger of the New_Exclusion_List and the Exclusion_List of the Recurring Master (step 563). Also a new FIG is created between the Synthetic Master and the CIG-mates of all FIG records from the History File (step 565).

In steps 567-569, the Synchronizer checks to see if there are some instances of the Recurring Master which fall within the previous synchronization's date range but fall outside of the current synchronization's date range. If so, the Fan_Out_Creep flag is set, indicating that the date range has moved in such a way as to require the record to be fanned for the database before unloading the record. The Fan_Out_Creep flag is an increase in the value in the Non_Key_Field Hash of the Recurring Master. In this way, the Recurring Master during the unloading of the records will appear as having been updated since the last synchronization and therefore will be fanned for the current date range.

In step 570, all the FIG records analyzed or created in this analysis are marked as Dependent_FIGs. This results in these records being ignored in future analysis except when the recurring records to which they are attached are being analyzed.

At the end of the above analysis, all the records having a unique ID assigned by their databases have been matched based on their unique ID. From this point onward, the records which do not have unique IDs must be matched to other records based on their field values. In the pr ferr d embodiment, th r ar two cat g ries f fi ld valu matches: strong matches and w ak matches. A strong match betwe n two records that hav matching key fields is when non-k y fields of the two records match or it is a Recurring Master and a fanned instance match (Fig. 14, steps 606-610). Referring to Fig. 15, a weak match between two records that have matching key fields is when the following are true: each of the two records are from different origins, because two records from the same source should not be in a CIG (e.g., A_Database and History File); each is not a weak match for another record because there is no reason to prefer one weak match over another; each is not a Dependent_FIG since these records do not have an independant existence from their recurring masters; both records are either recurring or non-recurring since a recurring and a nonrecurring should not be matched except if one is an instance of the other in which case it is a strong match; and, in case of non-recurring, they have matching Key_Date_Field which is the same as the Start_Date in the preferred embodiment because items on the same date are more likely to be modified versions of one another.

Referring to Fig. 14, these two types of matching are used to match records to existing CIGs for History File records which have been created based on matching unique IDs. Only doubleton CIGs are looked at, because singleton CIGs are handled in step 504 of Fig. 12 and tripleton CIGs are complete (steps 601-604). If a strong match is found, then if the record was a weak match in another CIG, it is removed from that CIG, and new weak match is found for that CIG (612-614). While weak matches are left in SKGs in case they will find a strong match, strong matches are removed from their SKGs (step 614). If a strong match is not found, then a weak match is sought (steps 617-620). All records in the CIG are remov d from SKG if no weak match is found, becaus this means that there is no possibility of ev n a weak match for this r cord (step 619).

The next step in CAAR is finding non-unique ID bearing instances for recurring items (Fig. 12, step 503). Referring to Fig. 16, this analysis takes place only if the database from which instances matching a recurring record are sought does not provide unique ID or if we are synchronizing from scratch (steps 650-653). The goal of this analysis is to find matching instances for each Recurring Master from a different source than the Recurring Master. This analysis counts the number of records in SKG of the Recurring Master which have matching Non_Date_Hash value (steps 665-669). The group of matching SKG records having the same non_Date_Hash value and having the highest number of members (if the number of members exceeds 30% of unexcluded instances) is then formed into a Homogeneous_Instances_Group (steps 670-672). A Synthetic Master is created using the Rep_Basic of the Recurring Master and using the values from the homogeneous instances group. An Exclusion list is created based on the items belonging to the recurrence pattern but missing from the Homogeneous_Instances_Group. The Synthetic Master is added to the CIG of the Recurring Master (steps 673-678). A new FIG for the Synthetic Master is then created using the Homogeneous_Instances_Group (step 679). These records are removed from any CIGs to which they belonged as weak matches and new weak matches are sought for those CIGs (steps 680-684). Since the records in Homogeneous_Instances_Group have now been matched to a recurring record, they are marked as Dependent_FIGs (step 683). The Recurring Master's CIG is then marked with Fan_Out_Creep flag, if necessary (step 685).

The next step in CAAR is completing analysis of rec rds in SKGs (Fig. 12, step 504). Referring to Fig. 17, this analysis attempts to incr ase th populati n of CIGs up to a maximum by finding k y fi ld bas d matches with records from a source different from those of th CIG records. This analysis is p rform d by analyzing all the records in the SKGs except for the singleton SKGs (steps 703 and 712). The first thing is to remove any members that have already been marked as WEAK matches attached to ID-based doubleton CIGs. Those are left in the SKG up to this point to allow for the possibility that a STRONG match would be found instead. But that is not possible any longer (steps 713-715). Once the weak matches have been removed, all remaining SKG members belong to singleton CIGs. Any non-singleton CIGs which are formed from here on will be purely key field based.

Throughout the remaining SKG Analysis we are careful not to seek H_Record-A_Record or H_Record-B_Record matches for unique ID-bearing Source, since that would violate the exclusively ID-based matching scheme that applies in such cases. Note however that an A_Record-B_Record match is acceptable even if both A_Database and B_Database are unique ID-bearing databases.

Given that Key Field should not be performed where ID based matches are available (or otherwise there may be matches between records with differing IDs), there are limits to how big CIGs can get at this point. If both A and B_Databases are unique ID-bearing, any remaining H_Record must remain in Singleton CIGs, because they are prohibited from forming key fields based matches with items from either databases. Such H_Records are simply removed from the SKG when they are encountered. If just one of the two databases being synchronized is unique ID-bearing then the maximum population that any CIG can now attain is 2 (Fig. 18, steps 750-751). If neither database is unique ID bearing th n the CIG_Max_Size is three. F r v ry CIG which is analyzed in Fig. 17, th CIG_Max_Size is set according to this logic. Wh n a CIG reaches its maximum possibl population all of its memb rs are removed from th appropriate SKG.

First, strong matches for the H-records are searched for, before trying to find A-B matches. If both Databases are non-unique ID-bearing then two strong matches for each H_Record, an H-A and an H-B match, are sought (steps 715-720). If finding a strong match results in reaching the CIG_Max_Size, all members of the CIG are removed from the SKG (step 721).

When maximum CIG population is 3, weak matches are sought for strong matching CIG doubleton in order to build triplet CIGs. The first weakly matching SKG member is added to the CIG (steps 722-728). Whether or not a weak match is found for any of the doubleton CIGs, its members are removed from the SKG (step 726). As there are no strong matches left in the SKG, weak matches are found for any remaining SKG members and joined to them in CIGs (steps 722-725).

At this stage, all CIGs are built. They must now be examined to determine what needs to be done to these records so that the databases are synchronized, i.e. whether the records in the CIGs need to be added, deleted or changed in the two databases. First step is determining the CIG_TYPE which represents the relation between the records. The following CIG types are defined, all using a 3-digit number that represents values found for A_DATABASE, History File, and B_Database, respectively:

1. 001 - record is "new" in the B_DATABASE
2. 010 - record is present in History, but absent in both A_Database and B_Databases
3. 100 - record is "new" in the A_Database
4. 101 - record is "new" in both A_Database and B_DATABASE; same in both

5. 102 - r cord is "new" in both A_Database and B_DATABASE; different in each (conflict)
6. 110 - record deleted from B_DATABASE
7. 011 - record deleted from A_Database
8. 012 - record deleted from A_Database and changed on B_DATABASE (DEL vs CHANGE conflict)
9. 210 - record changed on A_Database and deleted from B_DATABASE(DEL vs CHANGE conflict)
10. 111 - record unchanged since previous synchronization
11. 112 - record changed on B_DATABASE only since previous synchronization
12. 211 - record changed on A_Database only since previous synchronization
13. 212 - record changed identically on both since previous synchronization
14. 213 - record changed differently on each since previous synchronization (conflict)
15. 132 - a conflict (102 or 213) was resolved by forming a compromise value; Update both
16. 13F - created when a 132 Update both CIG is Fanned into the B_DATABASE

Fig. 19 shows the method used for setting all except the last two CIG_Types which are set in other operations.

Four of the CIG types assigned above involve conflicts: 102, 213, 012, and 210. Conflicts are those instances where a specific conflict resolution rule chosen by the user or set by default, or the user's case by case decision, must be used to determine how the records from the databases should be synchronized. CIG types 012 and 210 are cases where a previously synchr niz d record is chang d on one sid and del ted on the other. In the preferr d mb diment, such conflicts are resolved acc rding t th rule that CHANGE overrules the DELETE. So th net r sult for CIG type 012 is to add a new record to the A_Database to match the record in the B_DATABASE. The reverse is true for CIG type 210, where a new record is added to the B_Database. In an alternate embodiment, the user may be allowed to register an automatic preference for how to resolve such conflicts or decide on a case-by-case basis a conflict resolution option.

The other two conflict types -- 102 and 213 -- are resolved in the preferred embodiment according to the Conflict Resolution Option established by the user. First, the user may choose to ignore the conflict. This option leaves all 102 and 213 conflicts unresolved. Every time synchronization is repeated the conflict will be detected again and ignored again, as long as this option remains in effect and as long as the conflicting records are not changed by other means.

The user may choose to add a new record to each of the two databases. This option resolves 102 and 213 conflicts by adding the new A_Record to the B_Database, and adding the new B_Record to the A_Database. This option is implemented by breaking a 102 CIG into two separate CIGs (types 100 and 001) and a 213 CIG into three separate CIGs (types 100, 010, and 001). Subsequent processing of those descendant CIGs causes new records to be added across and stored in the History File.

The user may elect that A_Database records should always trump or win over B_database records. This option is implemented by changing the CIG type to 211 - the processing during unloading the records changes the record value in the B_Database to match the current record value in the A_Database.

The user may el ct that B_Databas records should always trump or win over B_databas records. This option is implem nted by changing th CIG typ to 112 - the processing during unloading th records chang s th record value in the A_Database to match the current record value in the B_Database.

The user may choose to be notified in case of any conflict. The user is notified via a dialog box 30, shown in Fig. 20, whenever a CIG type conflict of 102 or 213 arises. The dialog box shows the record that is involved in the conflict 31. It also shows the A_Database 32 and B_Database 33 values for all conflicting fields, in a tabular display, with Field Names appearing in the left column 34. A dropdown list (not shown) in the lower left hand corner of the dialog 37, offers a total of three choices - add, ignore, and update. The use may choose to add new records or ignore the conflict. The user may also choose that the A_Record or B_Record should be used to update the other record. The user may also decide to create a compromise record by choosing values of different fields and then choosing update option. In this case, the CIG type is changed to 132, which results in an updating both databases with the new record compromise record.

When the user has chosen to be notified in case of conflict, if the user chooses to ignore conflict or that either the record of the A_Database or the B_DATABASE should win, the CIG type is left as a conflict CIG type (102 or 213) and a separate Conflict Resolution Choice is stored in the FLAGS word associated with each CIG member.

The final step in setting CIG_Types is the process for dealing with difficulties which arise from exclusion lists. For example, in a triple Recurring Master CIG, suppose the History File Recurring Master does not have any exclud d instances. Th A_R cord has the f llowing exclusion list:

12/1/96, 12/8/96

Th B_Rec rd has th following xclusion list:

1/1/97, 1/8/97, 1/15/97, 1/22/97, 1/29/97

If comparison of the Recurring Masters includes comparing exclusion list Field Values, this set of changes would cause the Synchronizer to report a CIG type 213 conflict.

If the Conflict Resolution Option is set to A_Database record wins, then the outcome prescribed by the Synchronizer would be for the A_Database to keep its exclusion list as is and for the B_Database to make its exclusion list match that of the A_Database.

The result would be to have a lot of duplicate entries in both Databases. The A_Database would have five duplicate entries in January 97 - that is the five unmodified Recurring Master instances, plus the five modified instances added across from B_Database to A_Database. The B_Database would have five duplicate entries in January 97, since synchronization has wiped out the five exclusions that were previously recorded in the B_Database exclusion list.

Two steps are implemented for dealing with this problem. First, the COMPARE function does not take into account exclusion list differences when comparing recurring records. Second, referring to Fig. 21, any new exclusions added on to one recurring record will be added to the other record. The merging of exclusion lists is done regardless of any updates or conflicts, even unresolved conflicts, between the A_Database and B_Database copies of a Recurring Master. One exception is for CIG type 102 conflict which is left unresolved where Exclusion lists are not merged, because the user has chosen to leave those records as they are.

In m st cas s wh r it is necessary to merge exclusion lists, the CIG typ s and/or the Conflict Resolution Ch ice to arrang for all n cessary updates to b performed during the unloading phases of synchronization.

First, A_Database and B_Database records' exclusion lists are compared. In case of databases which do not permit recurring items, the exclusion list of the Synthetic Master is compared to the recurring record of the other database (step 852). If there is no difference, then nothing is done (step 853). If there are differences, then it is determined which exclusions appear only in one record. This comparison always yields one of the following scenarios: (1) all one-side-only Exclusions are on the A_Database (so Exclusions should be added to the B_Database); (2) all one-side-only Exclusions are on the B_Database (so Exclusions should be added to the A_Database); and (3) there are one-side-only Exclusions on both sides (so Exclusions should be added to both databases).

In each of these cases a separate table is used to look up instructions, for how to handle each specific situation (Figs 22-24). The tables cover all possible combinations of previous CIG types and outcome codes with all possible exclusion list changes (new and different exclusions added on A_Database, or on B_Database, or on both sides). Fig. 22 table is used in case of scenario 1. Fig. 23 table is used in case of scenario 2. Fig. 24 table is used in case of scenario 3 (Fig. 21 steps 854-856).

The analysis of records is now complete, and the records can be unloaded into their respective databases, including any additions, updates, or deletions. However, prior to doing so, the user is asked to confirm proceeding with unloading (Fig. 3, step 108-109). Up to this p int, n ith r of the databases nor th History File have been modified. The user may obtain through th

Translation Engine's User Interface various information regarding what will transpire upon unl ading.

If the user chooses to proceed with synchronization and to unload, the records are then unloaded in order into the B_Database, the A_Database and the History File. The Unloader modules 6,10 of the Translators 5,9 perform the unloading for the databases. The Synchronizer creates the History File and unloads the records into it. The Control Module 2 of the Translation Engine 1 first instructs the B_Translator to unload the records from Workspace into the B_Database. Referring to Fig. 25, for each CIG to be unloaded (determined in steps 902-907), based on the CIG_TYPE and which database it is unloading into (i.e., A or B), the unloader looks up in the table in Fig. 26 the outcome that must be achieved by unloading - that is, whether to update, delete, add, or skip (Leave_Alone) (step 908). In steps 909-913, the unloader enforces date range restriction for a database subject to date range. The user may select, or a selection may be made by default, whether to enforce the date range sternly or leniently. In case of stern enforcement, all records outside of the current date range would be deleted. This is useful for computers with small storage capacity. In case of lenient enforcement, the records are left untouched.

Based on the result obtained from looking up the unloading outcome in the table, the unloader then either adds a new record (steps 920-926), deletes an existing record (steps 914-919), or updates an existing record (steps 927-933). It should be noted that because we only update those fields which need to be updated (step 928), the fields which were sanitized but need not be updated are n t unloaded. Therefore, the values in those fi lds remain in unsanitized form in the database.

Ref rring to step 914, in some Applications when a Recurring Master must b added or updated, the r cord may have to be fanned out despite the ability of the Application to support recurring records. For example, the Schedule+ Translator is generally able to put almost any Recurring Master Item into Schedule+ without fanning, but there are some exceptions. The Schedule+ Translator uses one Schedule section to handle all appointments and events. For appointments, almost any recurrence pattern is allowed, but for events the only allowable true repeat type is YEARLY. DAILY recurring events can be dealt with by being translated into Schedule+ multi-day events which are not recurring but extend over several days by setting the EndDate some time after the Start Date. But for the DAILY case there are restrictions. In particular exclusions in the midst of a multi-day Schedule+ event cannot be created. So the Translator decides that if section type is ToDos or the item is a non-Event Appointment, then the record need not be fanned out. But if item is a YEARLY or DAILY with no exclusions then it can be stored as a Schedule+ yearly or daily event. Otherwise, it must be fanned.

Referring to Fig. 27, steps 950-984 set out the preferred embodiment of fanning recurring records that must be updated. All cases fall within three scenarios, shown in Fig. 29.

In the first scenario a record which is a Recurring Master, and its counterpart in the other database is a Recurring Master, must be fanned now for its own database (steps 951-959). If the CIG_TYPE of the record is 132 (i.e. update both records), then it is changed to 13F which is a special value specifically for this situation (step 951). For other CIG_Types, the CIG is brok n into thre singl ton and given CIG_Types signifying th ir singleton status. In both f these cas s, th function Fanning_For_Add (steps 986-996, describ d b low) is called.

In the second scenario, the record was fanned previously and is going to be fanned now also. First, the dates of the instances are recorded in a temporary date array (steps 961-963). This array is compared to an array of the fanned instances of the recurrence pattern of the CIG Recurring Master from the other database (steps 965-966). The dates which are not in the array of fanned instance are marked for deletion (step 967). The dates which are not in the temporary date array should be added to the unloading databases and therefore new FIG records are created for those dates (steps 968-973). The dates which appear in both arrays are compared to the Synthetic Master and marked accordingly for UPDATE or Leave_Alone (steps 974-978).

In the third scenario, the record which was previously fanned should now be fanned also. The opposing database's record in this scenario is also fanned instances. This is perhaps the most peculiar of the three cases. For example, a database may be able to handle multi-day (i.e. daily recurring) records but not any exclusion dates for such items. Such database may be synchronized with another database which fans all records in the following manner. A record representing a 7-day vacation in the Planner section of the database is fanned out to form 7 individual vacation days in the other database. One instance is deleted in the other database. Upon synchronizing the two databases, b/c the first databases does not does not provide for exclusion lists, the record must now be fanned.

In this scenario, Master Records in a CIG are mark d as Garbage. Any FIG members attached to the H_Record, if any, are also marked as Garbage. All Instanc s found in the opposing database's FIG ar truned to singleton CIGs with CIG type 100 or 001 s that th y will be add d to the unloader's database wh n unloading is done. In this way the instances from one database is copied to the database providing for recurring records.

Steps 985-995 describe the Fanning_For_Add Function which is used when outcome is to update or when the function is called by the Translator fanning for update. For each instance generated by fanning out the recurring record, a clone of the Recurring Master is created but excluding Rep_Basic and Rep_Excl field values and the unique ID field. All adjustable Date Fields (e.g. Start Date, End Date, and Alarm Date) are set and hash values for the new record is computed. The new record is then marked as Fanned_For_A or Fanned_For_B, as the case may be. This is then attached to the Recurring Master Item as a FIG member.

Following unloading of the B_RECORDS, the Control Module 2 instructs the A_Translator to unload the A_Records from the Workspace (Fig. 3, step 111). This unloading is done in the same way as it was done by the B_Translator. In case of Rebuild_All Translators which have to reconstruct the database, all records which were loaded from the database but were not used in synchronization are appended and unloaded as the Translator builds a new database for its Application.

The Control Module 3 next instructs the Synchronizer to create a new History File (step 112). Referring to Fig. 28, for every CIG in the Workspace, it is first determined which record should be unloaded to History File (steps 1001-1003). In the next step, Excl_Only flag is checked, which is set by the Merge_Exclusion_List logic (Fig. 21-24). If that flag is s t, a n w r cord for unloading is created which has all fields taken from the History File record, except that the newly m rged exclusion list is inserted into that rec rd (step 1004). Before storing th r cord in th Hist ry File, all Flag Bits in th Extended Ind x are cleared except the bit that indicating whether or not this is a recurring item (step 1005). The item is marked as a History File record to indicate its source. The CIG, FIG, and SKG are reset. All the HASH values and Start&EndDate&Time will be stored. All applicable unique ID are also stored (Steps 1006-1009). The current record is then stored in the new History File (step 1010). If the current record is a Recurring Master for an ID-bearing FIG, we now store the whole FIG (i.e. all Fanned Instances) in the History File, with the FIG linkage words set in the History File to hold the FIG records together (step 1011). Fanned instances which do not bear unique IDs are not stored in the History File since they can be re-generated by merely fanning out the Recurring Master.

Once all records are unloaded, various information necessary for identifying this History File and for the next synchronization are written into the History File (step 1013).

At this point Synchronization is complete.

Applications, such as scheduling Applications, often have more than one database. Each of these databases are known as sections. Each of these sections contain different data and must be synchronized with their corresponding sections in other Applications. However, there is not necessarily a one to one relationship between sections of various Applications. For example, Application A may comprise of the following sections: Appointments, Holidays, Business Addresses, Personal Addresses, and ToDo. Application B however may comprise of the following s ctions: Appointments, Address s, ToDo-Tasks, and ToDo-Calls. Although th general character f the sections are the same, ther is not a on to on relation b tw en the sections of thes two Applications: Appointments and Holidays in A contain the same type of data as Appointments in B; Business Addresses and Personal Addresses in A contain the same type of data as Addresses in B; and ToDo in A contains the same type of data as ToDo-Tasks and ToDo-Calls in B. Therefore, when synchronizing the sections of these two Applications, it is necessary to synchronize at least two sections of one Application with one section of another Application.

The preferred embodiment performs this type of synchronization by providing for a number of section categories: Appointment, ToDo, Note, Address, and General Database. All sections of a particular Application are studied and categorized according to this categorization. Therefore, in the above example of Application A, Appointments and Holidays are categorized Appointment type sections (or database), Business Address and Personal Address as Address type sections, and ToDo as a ToDo type section.

For creating the map for mapping sections onto each other, an exact section match is always sought between sections of the two Applications. If not, one of the sections which were categorized as a section type is chosen to be the Main_Section among them. Other sections of the same type are referred to as subsections. All databases of the same type from the other Application will be mapped onto the Main_Section.

To properly synchronize from one time to the next, it is necessary to keep track of the source of records in the Main_Section. In the preferred embodiment, if a record in the Main_Section of the A_Application does not come from th Main_S ction of the B_Application, n f fields in the record, pr f rably a text fi ld, is tagged with a uniqu code id ntifying the subs ction which is th sourc of the record. This is the record's Origin Tag. All r cords in th Workspace and the History Fil include a hidden internal field called _subType which contains the unique subsection code. Main_Section's field value in the preferred embodiment is zero so that it will not be tagged. When a record is loaded from a database into the Synchronization Workspace, the tag is stripped from the TagBearer field and put in the _subType field. If there is no tag, then the _subType is set to be the subType of the present section. If the TagBearer field is mapped then when reading records into the Workspace the tag, if any, is stripped from the TagBearer field value place it in _subtype.

Conversely when unloading records from the Workspace to a Database, the TagBearer field is tagged by a tag being added if the record is not from the Main_Section.

Other embodiments are within the following claims.

What is claimed is:

1. A computer implemented method of synchronizing at least a first and a second database, wherein the manner of storing a set of recurring date bearing instances differs between the first and second databases, and at least the first database uses a recurring record to store the set of recurring date bearing instances, the method comprising:

processing a plurality of non-recurring records in the second database to generate a synthetic recurring record representing a set of recurring date bearing instances in the second database;

performing a comparison of the synthetic recurring record of the second database to a recurring record of the first database;

completing synchronization based on the outcome of the comparison.

2. The method of claim 1 wherein the step of completing synchronization includes adding, modifying, or deleting one of the synthetic recurring record and recurring record.

3. The method of claim 1 wherein, following the step of completing synchronization, one of the synthetic recurring record and recurring record is fanned back into a plurality of fanned non-recurring records.

4. The method of claim 1 wherein the set of recurring date bearing instances is stored in the second database as a plurality of non-recurring records.

5. The method of claim 1 wh rein th set of recurring date bearing instances is stor d in the s cond database as a recurring record having a different record structure than the recurring record of the first database.

6. The method of claim 1 further comprising storing a history file containing a record representative of one of the recurring record and synthetic recurring record in a past synchronization.

7. The method of claim 1 wherein the synthetic recurring record has a list of excluded instances and the step of processing a plurality of non-recurring records in the second database to generate a synthetic recurring record further comprises generating a list of excluded instances representative of instances previously represented by the recurring record and currently represented by another record or deleted.

8. The method of claim 1 wherein the recurring record and the synthetic recurring record each contain a list of excluded date bearing instances, wherein the step of performing a comparison of the synthetic recurring record to the recurring record includes performing a comparison of the list of excluded date bearing instances of the recurring record with the list of excluded date bearing instances of the synthetic recurring record.

9. The method of claim 8 wherein the step of completing synchronization includes adding, modifying, or deleting the list of excluded date bearing instanc s of one f the recurring record and th synthetic recurring rec rd.

10. The method of claim 8 wh r in the step of completing synchronization includes adding, modifying, or deleting one of the synthetic recurring record and recurring record.

11. The method of claim 8 wherein, following the step of completing synchronization, one of the synthetic recurring record and recurring record is fanned into a plurality of fanned non-recurring records excluding the instances in the list of excluded date bearing instances of a corresponding one of the synthetic recurring record and recurring record.

12. The method of claim 6 wherein the second database assigns a unique ID to each record, and wherein the method further comprises:

fanning one of the synthetic recurring record and the recurring record into a plurality of fanned non-recurring records;

storing records in the history file representative of the plurality of fanned non-recurring records;

storing in the history file the unique IDs assigned by the second database to the plurality of fanned non-recurring records; and

recording linkages among the records representative of the plurality of non-recurring records and the record representative of one of the recurring record and synthetic recurring record.

13. The method of claim 6 wherein the second database assigns uniqu IDs to each r cord, the history file further contains records representative of non-recurring records of the second database from a past synchronization and unique IDs assigned to the non-recurring records of the second database, and the step of processing a plurality of non-recurring records in the second database to generate a synthetic recurring record further comprises:

performing a comparison of the unique IDs stored in the history file with unique IDs of the plurality of non-recurring records in the second database; and

selecting a set of non-recurring records in the second database based on the comparison of the unique IDs and generating the synthetic recurring record using the set of non-recurring records.

14. The method of claim 13 wherein the step of selecting a set of non-recurring records further comprises selecting a set of non-recurring records in the second database having unique IDs matching a set of the unique IDs stored in the history file.

15. The method of claim 13 wherein one of the synthetic recurring record and the recurring record has an exclusion list and the step of selecting the set of non-recurring records comprises:

selecting a set of records in the history file having unique IDs failing to match any of the unique IDs of non-recurring records in the second database; and

adding, modifying, or deleting the exclusion list of at least one of the synthetic recurring record and the recurring record, using the set of records in th history file.

16. Th m thod of claim 6 further comprises p rforming a second comparison of one of the synthetic recurring record and the recurring record to the record representative of the recurring record or the synthetic recurring record in a past synchronization, and completing synchronization based on the outcome of the second comparison.

17. The method of claim 1 wherein each recurring record and each non-recurring record includes a key field, and wherein the step of processing a plurality of non-recurring records in the second database to generate the synthetic recurring record further comprises:

performing a second comparison of the key fields of the recurring and non-recurring records; and

selecting a group of records from among the recurring and non-recurring records based on the outcome of the comparison.

18. The method of claim 17 wherein the step of selecting a group of records comprises selecting the group based on identity of the content of the key fields of the recurring and non-recurring records.

19. The m thod of claim 17 wherein each recurring r cord and each non-recurring record includes at least one other field, and wherein the step of processing a plurality of non-recurring records in the second database to generate a synthetic recurring record further comprises:

performing a third comparison of the at least one other field of the non-recurring records in the group;

selecting a set of non-recurring records based on the outcome of the third comparison; and

generating the synthetic recurring record using the set of non-recurring records.

20. The method of claim 19 wherein selecting the set of non-recurring records based on the outcome of the third comparison is based on identity of content of the at least one other field of the non-recurring records in the group.

21. A computer program, r sident on a computer readable medium, for synchronizing at 1 ast a first and a second database, wherein the manner of storing a set of recurring date bearing instances differs between the first and second databases, and at least the first database uses a recurring record to store the set of recurring date bearing instances, comprising instructions for:

processing a plurality of non-recurring records in the second database to generate a synthetic recurring record representing a set of recurring date bearing instances in the second database;

performing a comparison of the synthetic recurring record of the second database to a recurring record of the first database;

completing synchronization based on the outcome of the comparison.

22. The computer program of claim 21 wherein the instruction for completing synchronization includes adding, modifying, or deleting one of the synthetic recurring record and [or the] recurring record.

23. The computer program of claim 21 wherein, following the instruction for completing synchronization, one of the synthetic recurring record and recurring record is fanned back into a plurality of fanned non-recurring records.

24. The computer program of claim 21 wherein the set of recurring date bearing instances is stored in the second database as a plurality of non-recurring r cords.

25. The computer program of claim 21 wherein th set f recurring date bearing instances is stored in the second database as a recurring record having a different record structure than the recurring record of the first database.

26. The computer program of claim 21 further comprising instructions for storing a history file containing a record representative of one of the recurring record and synthetic recurring record in past synchronization.

27. The computer program of claim 21 wherein the synthetic recurring record has a list of excluded instances and the instruction for processing a plurality of non-recurring records in the second database to generate a synthetic recurring record further comprises instructions for generating a list of excluded instances representative of instances previously represented by the recurring record and currently represented by another record or deleted.

28. The computer program of claim 21 wherein the recurring record and the synthetic recurring record each contain a list of excluded date bearing instances, wherein the instruction for performing a comparison of the synthetic recurring record to the recurring record includes instructions for performing a comparison of the list of excluded date bearing instances of the recurring record with the list of excluded date bearing instances of the synthetic recurring record.

29. Th computer program of claim 28 wher in th instruction for completing synchronization includes instructions for adding, modifying, or deleting the list of excluded date bearing instances of one of the recurring record and the synthetic recurring record.

30. The computer program of claim 28 wherein the instruction for completing synchronization includes instructions for adding, modifying, or deleting one of the synthetic recurring record and recurring record.

31. The computer program of claim 28 wherein, following the instruction for completing synchronization, one of the synthetic recurring record and recurring record is fanned into a plurality of fanned non-recurring records excluding the instances in the list of excluded date bearing instances of a corresponding one of the synthetic recurring record and recurring record.

32. The computer program of claim 26 wherein the second database assigns a unique ID to each record, and wherein the computer program comprises:

fanning one of the synthetic recurring record and the recurring record into a plurality of fanned non-recurring records;

storing records in the history file representative of the plurality of fanned non-recurring records;

storing in the history file the unique IDs assigned by the second database to the plurality of fanned non-recurring records; and

r cording linkages among the r cords r presentative of th plurality of non-r curring records and th r cord representative of one of the recurring rec rd and synthetic r curring r cord.

33. The comput r program of claim 26 wh rein the s cond database assigns unique IDs to each r cord, th history file further contains records representative of non-recurring records of the second database from a past synchronization and unique IDs assigned to the non-recurring records of the second database, and the instruction for processing a plurality of non-recurring records in the second database to generate a synthetic recurring record further comprises instructions for :

performing a comparison of the unique IDs stored in the history file with unique IDs of the plurality of non-recurring records in the second database; and

selecting a set of non-recurring records in the second database based on the comparison of the unique IDs and generating the synthetic recurring record using the set of non-recurring records.

34. The computer program of claim 27 wherein the instruction for selecting a set of non-recurring records further comprises instructions for selecting a set of non-recurring records in the second database having unique IDs matching a set of the unique IDs stored in the history file.

35. Th comput r program of claim 33 wherein one of th synthetic recurring record and the recurring record has an exclusion list and the instruction for selecting the set of non-recurring records comprises instructions for :

selecting a set of records in the history file having unique IDs failing to match any of the unique IDs of non-recurring records in the second database; and

adding, modifying, or deleting the exclusion list of at least one of the synthetic recurring record and the recurring record, using the set of records in the history file.

36. The computer program of claim 26 further comprises instructions for performing a second comparison of one of the synthetic recurring record and the recurring record to the record representative of the recurring record or the synthetic recurring record in a past synchronization, and completing synchronization based on the outcome of the second comparison.

37. The computer program of claim 21 wherein each recurring record and each non-recurring record includes a key field, and wherein the instruction for processing a plurality of non-recurring records in the second database to generate the synthetic recurring record further comprises instructions for:

performing a second comparison of the key fields of the recurring and non-recurring records; and

selecting a group of records from among the recurring and non-recurring records based on the outcome of th comparis n.

38. The comput r program of claim 37 wherein the instruction for selecting a group of records comprises instructions for selecting the group based on identity of the content of the key fields of the recurring and non-recurring records.

39. The computer program of claim 38 wherein each recurring record and each non-recurring record includes at least one other field, and wherein the instruction for processing a plurality of non-recurring records in the second database to generate a synthetic recurring record further comprises instruction for:

performing a third comparison of the at least one other field of the non-recurring records in the group;

selecting a set of non-recurring records based on the outcome of the third comparison; and

generating the synthetic recurring record using the set of non-recurring records.

40. The computer program of claim 39 wherein selecting the set of non-recurring records based on the outcome of the third comparison is based on identity of content of the at least one other field of the non-recurring records in the group.

41. A computer program, resident on a computer readable medium, for synchronizing at least a first and a second database, wherein each record in the first and second databases includes a key field, comprising instructions for:

performing a first comparison of the content of the key field of records of the first database with the content of the key field of records of the second database;

selecting a plurality of groups of records of the first and second databases based on the outcome of the first comparison;

performing a second comparison of records in one of the plurality of groups of records; and

completing the synchronization based on the outcome of the second comparison.

42. The computer program of claim 41, the computer program further comprises instructions for selecting the plurality of groups of records based on identity of the contents of the key fields of the records of the first and second database.

43. The computer program of claim 41 wherein the instruction for completing synchronization further comprises instructions for selecting a corresponding item group of records based on the outcome of the second comparison wherein a corresponding item group of records comprises at least a record from one of the first and the second database.

44. Th c mputer program of claim 43 wh rein the instruction for completing synchronization further comprises instructions for:

performing a third comparison of the records of the corresponding item group; and

completing synchronization based on the third comparison.

45. The computer program of claim 43 further comprising instructions for storing a history file containing history records representative of records of the first and second databases in a past synchronization and wherein a corresponding item group further comprises a history record.

46. The computer program of claim 45 wherein the instruction for completing synchronization further comprises instructions for:

performing a third comparison of the records of the corresponding item group; and

completing synchronization based on the third comparison.

47. The computer program of claim 46 wherein the key field is a date field.

48. The computer program of claim 41 wherein the key field is a text field.

49. A computer implement d method of synchr nizing at l ast a first and a second databas , wherein each record in the first and second databases includes a key field, the method comprising:

performing a first comparison of the content of the key field of records of the first database with the content of the key field of records of the second database;

selecting a plurality of groups of records of the first and second databases based on the outcome of the first comparison;

performing a second comparison of records in one of the plurality of groups of records; and

completing the synchronization based on the outcome of the second comparison.

50. The method of claim 49, the method further comprises selecting the plurality of groups of records based on identity of the contents of the key fields of the records of the first and second database.

51. The method of claim 50 wherein the step of completing synchronization further comprises selecting a corresponding item group of records based on the outcome of the second comparison wherein a corresponding item group of records comprises at least a record from one of the first and the second database.

52. The method of claim 51 wherein the step of completing synchronization further comprises:

performing a third comparison of the records of the corr sponding item group; and

completing synchronization based on the third comparis n.

53. Th method of claim 52 further comprising storing a history file containing history records representative of records of the first and second databases in a past synchronization and wherein a corresponding item group further comprises a history record.

54. The method of claim 53 wherein the step of completing synchronization further comprises:

performing a third comparison of the records of the corresponding item group; and

completing synchronization based on the third comparison.

55. The method of claim 49 wherein the key field is a date field.

56. The method of claim 49 wherein the key field is a text field.

57. A computer implemented method of synchronizing records of first and second databases, wherein at least one field of records of the first database is subject to a first rule of data value to which the corresponding field of records of the second database is not subject, the method comprising:

comparing the content of the one field to the content of the corresponding field of the second database and in performing the comparison applying the first rule of data value;

taking synchronization actions based on the outcom of the comparison.

58. The method of claim 57 wher in at least ne field of records of the second database is subject to a second rule of data value to which a corresponding field of records of the first database is not subject, wherein the method further comprises applying the second rule of data value in performing a comparison of the content of the corresponding field of records of the first database to the content of the at least one field of the second database.

59. The method of claim 57 wherein applying the first rule of data value comprises:

using the first rule of data value to modify a corresponding field of records representative of the records of the second database; and

thereafter comparing the content of the modified corresponding field of the representative records to the content of the one field.

60. The method of claim 57 wherein the content of the one field comprises at least a first portion and a second portion and the first rule of data value requires the presence of the second portion, and wherein applying the first rule of data value comprises comparing only the first portion to the content of the corresponding field.

61. The method of claim 57 wherein the content of the corresponding field comprises at least a first portion and a second portion and the first rule of data value prohibits the content of the one field from containing the second portion and wherein applying the first rule f data value comprises comparing nly a first porti n of the content of th corresponding field to th one field.

62. The method of claim 57 wherein the first rule of data valu requires the content of the on field f the first database to have a specified value and wherein applying the first rule of data value comprises omitting comparison of the content of the one field with the content of the corresponding field.

63. The method of claim 57 wherein the first rule of data value limits the content of the one field to a first specified value and wherein applying the first rule of data value comprises setting the first specified value equivalent to a second specified value of the content of the corresponding field.

64. The method in claim 63 wherein the first specified value comprises a value selected from a range of values.

65. The method in claim 63 wherein the second specified value comprises a value selected from a range of values.

66. The method of claim 57 wherein applying the first rule of data value consists of one of:

a) comparing only a portion of the content of the one field to the content of the corresponding field;

b) comparing only a portion of the content of the corresponding field to the content of the one field;

c) omitting comparison of the content of the one field with the content of the corresponding field;

d) setting a first specified value of the one field quivalent to a second specified value of the corresponding field.

67. The method of claim 57 wherein the first rul of data value consists of one of:

a requirement that the content of the one field be in upper case;

a requirement that the content of the one field have a specified form of punctuation;

a requirement that the content of the one field have a specified form of spacing;

a requirement that the content of the one field have a value limited to a specified range of values;

a requirement that the content of the one field have a first specified value based on the content of another field;

a requirement that the content of the one field be limited to a specified length; and

a requirement that the content of the one field include a specified code.

68. A computer program, resident on a computer readable medium, for synchronizing records of first and second databases, wherein at least one field of records of the first database is subject to a first rule of data value to which the corresponding field of records of the second database is not subject, comprising instructions for:

comparing the content of the one field to the content of the corresponding field of the second database and in performing the comparison applying the first rule of data value;

taking synchronization actions based on the outcome of the comparison.

69. Th computer program of claim 68 wher in at l ast one field of r cords of the sec nd databas is subject to a second rule of data value to which a corresponding field of records of the first database is not subject, wherein the computer program further comprises instructions for applying the second rule of data value in performing a comparison of the content of the corresponding field of records of the first database to the content of the at least one field of the second database.

70. The computer program of claim 68 wherein applying the first rule of data value comprises:

using the first rule of data value to modify a corresponding field of records representative of the records of the second database; and

thereafter comparing the content of the modified corresponding field of the representative records to the content of the one field.

71. The computer program of claim 68 wherein the content of the one field comprises at least a first portion and a second portion and the first rule of data value requires the presence of the second portion, and wherein applying the first rule of data value comprises comparing only the first portion to the content of the corresponding field.

72. The comput r program of claim 68 wherein the content of th c rresponding field compris s at least a first portion and a second portion and the first rule of data value prohibits the content of the one field from containing the second portion and wherein applying the first rule of data value comprises comparing only a first portion of the content of the corresponding field to the one field.

73. The computer program of claim 68 wherein the first rule of data value requires the content of the one field of the first database to have a specified value and wherein applying the first rule of data value comprises omitting comparison of the content of the one field with the content of the corresponding field.

74. The computer program of claim 69 wherein the first rule of data value limits the content of the one field to a first specified value and wherein applying the first rule of data value comprises setting the first specified value equivalent to a second specified value of the content of the corresponding field.

75. The computer program of claim 74 wherein the first specified value comprises a value selected from a range of values.

76. The computer program of claim 74 wherein the second specified value comprises a value selected from a range of values.

77. The computer pr gram of claim 68 wh r in applying th first rule f data value consists of one f:

a) comparing only a portion of the content of the one field to the content of the corresponding field;

b) comparing only a portion of the content of the corresponding field to the content of the one field;

c) omitting comparison of the content of the one field with the content of the corresponding field;

d) setting a first specified value of the one field equivalent to a second specified value of the corresponding field.

78. The computer program of claim 68 wherein the first rule of data value consists of one of:

a requirement that the content of the one field be in upper case;

a requirement that the content of the one field have a specified form of punctuation;

a requirement that the content of the one field have a specified form of spacing;

a requirement that the content of the one field have a value limited to a specified range of values;

a requirement that the content of the one field have a first specified value based on the content of another field;

a requirement that the content of the one field be limited to a specified length; and

a requirement that the content of the one field include a specified code.

79. A computer implemented meth d of synchronizing at least a first and a second database with a third database, the method comprising:

selecting a first and a second set of records of the third database to be synchronized with records of a corresponding one of the first and second databases; and

synchronizing the first and second sets of records with the corresponding one of the first and second databases.

80. The method of claim 79, wherein the first set of records comprises records of the third database that were synchronized with records of the first database in a previous synchronization.

81. The method of claim 80 further comprising, in the previous synchronization, tagging the records of the third database that were synchronized with the records of the first database with an origin identification code identifying the source of the records as the first database.

82. The method of claim 81 wherein the records of third database added in the previous synchronization are tagged with the origin identifying code.

83. The method of claim 81 wherein the selecting step comprises selecting the records of the third database tagged with the origin identifying code as the first set of records.

84. The m thod of claim 81 wherein the selecting step comprises selecting r cords of the third database not tagg d with the origin id ntifying c de as the sec nd set of records.

85. A comput r program, resident on a computer readabl medium, for synchronizing at least a first and a second database with a third database, comprising instructions for:

selecting a first and a second set of records of the third database to be synchronized with records of a corresponding one of the first and second databases; and

synchronizing the first and second sets of records with the corresponding one of the first and second databases.

86. The computer program of claim 85, wherein the first set of records comprises records of the third database that were synchronized with records of the first database in a previous synchronization.

87. The computer program of claim 86 further comprising instructions for, in the previous synchronization, tagging the records of the third database that were synchronized with the records of the first database with an origin identification code identifying the source of the records as the first database.

88. The computer program of claim 87 wherein the records of third database added in the previous synchronization are tagged with the origin identifying code.

89. The computer program of claim 87 wherein the selecting instruction comprises instruction for selecting the records of th third database tagg d with the origin identifying code as th first s t of r cords.

90. The computer program of claim 87 wherein th s l cting instructions comprises instructi n f r selecting records of the third database not tagged with the origin identifying code as the second set of records.

91. A computer implemented method of synchronizing at least a first and a second database each containing dated records such as events, wherein the records of the first database extend across a narrow date range narrower than the date range of the records of the second database, the method comprising:

performing a prior synchronization across a prior date range set using the date of the prior synchronization and the narrow date range;

storing the prior date range and a history file containing information representative of the content of the databases following the prior synchronization;

performing a current synchronization across a date range that combines the prior date range with a current date range set using the date of the current synchronization and the narrow date range.

92. The method of claim 91 wherein the date of each record being synchronized is compared to a start and stop date of a date range to determine whether the record is in range.

93. A computer program, resid nt on a computer readable medium, for synchronizing at least a first and a second database each containing dated records such as events, wherein the records of the first database extend across a narrow date range narrower than the date range of the records of the second database, comprising instructions for:

performing a prior synchronization across a prior date range set using the date of the prior synchronization and the narrow date range;

storing the prior date range and a history file containing information representative of the content of the databases following the prior synchronization;

performing a current synchronization across a date range that combines the prior date range with a current date range set using the date of the current synchronization and the narrow date range.

94. The computer program of claim 93 wherein the date of each record being synchronized is compared to a start and stop date of a date range to determine whether the record is in range.

95. A computer implemented method of synchronizing at least a first and a second database, each one containing date bearing records, the method comprising:

identifying date bearing records of the first and second database that are within a narrow date range narrower than a date range of the records of one of the first and the second databases; and

performing a current synchronization across the narr w date range by synchronizing th id ntifi d dat b aring records.

96. Th method of claim 95 wherein the st p of performing a current synchronization further comprises adding, modifying, or deleting records of the first database within the narrow date range.

97. The method of claim 95 wherein the step of performing a current synchronization further includes deleting records of the first database that are outside of the narrow date range.

98. The method of claim 95 wherein the narrow date range has a start date and a stop date and the date of a record being synchronized is compared to the start date and the stop date of the narrow date range to determine whether the record is within the narrow date range.

99. The method of claim 98 wherein the date of a record being synchronized includes a record start date and a record stop date, the method further comprising:

performing a comparison of the record start date to the stop date of the narrow date range and of the record stop date to the start date of the narrow date range;

determining based on the comparison whether the record is within the narrow date range.

100. The method of claim 95 wherein the first database contains a recurring record, the method further comprising fanning the recurring record within the narrow date range.

101. The method of claim 95 wh rein th first database contains a r curring record, furth r comprising:

fanning the recurring record within a useful portion of the narrow date range, the useful portion being determined by application of a preference based on a current date of the current synchronization.

102. The method of claim 101 wherein the preference includes a preference for future dates compared to the current date over past dates compared to the current date.

103. The method of claim 101 wherein the preference includes a preference for dates closer to the current date over dates further from the current date.

104. The method of claim 95 wherein a prior synchronization was performed across a prior narrow date range, such prior narrow date range being different from a current narrow date range of the current synchronization, wherein records representatives of the records of the first and second databases during the prior synchronization are stored in a history file, and wherein performing the current synchronization further comprises performing the synchronization using the history file and the prior narrow date range.

105. The method of claim 104 wherein the narrow date range is a concatenation of the current narrow date range and the prior narrow date range stored in a history file during the prior synchronization.

106. Th m thod of claim 104 wher in th st p of perf rming a current synchronization further c mprises adding, modifying, or deleting records of the first database that are within the current narrow date range.

107. The method of claim 104 wherein the step of performing a current synchronization further includes deleting records of the first database that were present during a previous synchronization and that, following the current synchronization, are outside of the current narrow date range.

108. The method of claim 104 wherein the step of performing a current synchronization further includes updating or deleting records of the first and second databases that are outside of the current narrow date range and within the narrow date range, based on the current synchronization.

109. The method of claim 104 wherein the step of performing a current synchronization further comprises, based on a selection by a user, performing one of:

(a) deleting the records of the first database that were present during a previous synchronization and that, following the current synchronization, are outside of the current narrow date range, and

(b) updating or deleting records of the first and second databases that are outside of the current narrow date range and within the narrow date range, based on the current synchronization.

110. The meth d of claim 104 wherein one of the narrow date range, th prior narrow date range, and the current narrow date range includes a start date and a stop date and the date of a record being synchronized is compared to the start date and the stop date to determine whether the record is within a corresponding one of the narrow date range, the prior narrow date range, and the current narrow date range.

111. The method of claim 110 wherein a record being synchronized includes a record start date and a record stop date, the method further comprising:

performing a comparison of the record start date to the stop date of one of the narrow date range, the prior narrow date range, and the current narrow date range, and of the record stop date to the start date of the corresponding one of the narrow date range, the prior narrow date range, and the current narrow date range; and

determining based on the comparison whether the record is within the corresponding one of the narrow date range, the prior narrow date range, and the current narrow date range.

112. The method of claim 95 wherein the narrow date range comprises a relative narrow date range, the relative narrow date range being determined relative to a date of the current synchronization.

113. The method of claim 104 wherein the current narrow date range comprises a relative narrow date range, the relative narrow date range being determin d relative to a date of the current synchr nization.

114. The method of claim 95 wh rein th first database contains a first plurality of non-r curring records representing a plurality of recurring date bearing instances, the method further comprising:

generating a synthetic recurring record using the first plurality of non-recurring records;

performing a synchronization of the synthetic recurring record with a record of the second database;

if one of the record, the synthetic record, and a non-recurring record in the first plurality of non-recurring records is outside the narrow date range, fanning the synthetic record, or the record of the second database if it is a recurring record, into a second plurality of non-recurring records within the narrow date range.

115. The method of claim 95 wherein the narrow date range comprises a concatenation of a first date range for the first database and a second date range for the second database.

116. A computer program, resident on a computer readable medium, for synchronizing at least a first and a second database, each one containing date bearing records, comprising instructions for:

identifying date bearing records of the first and second database that are within a narrow date range narrower than a date range of the records of one of the first and the second databases; and

performing a current synchronization across the narrow date range by synchronizing the identified date bearing records.

117. The computer program of claim 116 wher in the instruction for performing a current synchronizati n further comprises instructions for adding, modifying, or deleting records of the first database within the narrow date range.

118. The computer program of claim 116 wherein the instruction for performing a current synchronization further includes instructions for deleting records of the first database that are outside of the narrow date range.

119. The computer program of claim 116 wherein the narrow date range has a start date and a stop date and the date of a record being synchronized is compared to the start date and the stop date of the narrow date range to determine whether the record is within the narrow date range.

120. The computer program of claim 119 wherein the date of a record being synchronized includes a record start date and a record stop date, the computer program further comprises instructions for:

performing a comparison of the record start date to the stop date of the narrow date range and of the record stop date to the start date of the narrow date range;

determining based on the comparison whether the record is within the narrow date range.

121. The computer program of claim 116 wherein the first database contains a recurring record, the computer pr gram further comprising instructions f r fanning the recurring record within the narrow dat rang .

122. The comput r program of claim 116 wher in the first database contains a recurring record, further comprising instructions for:

fanning the recurring record within a useful portion of the narrow date range, the useful portion being determined by application of a preference based on a current date of the current synchronization.

123. The computer program of claim 122 wherein the preference includes a preference for future dates compared to the current date over past dates compared to the current date.

124. The computer program of claim 122 wherein the preference includes a preference for dates closer to the current date over dates further from the current date.

125. The computer program of claim 116 wherein a prior synchronization was performed across a prior narrow date range, such prior narrow date range being different from a current narrow date range of the current synchronization, wherein records representatives of the records of the first and second databases during the prior synchronization are stored in a history file, and wherein performing the current synchronization further comprises instructions for performing the synchronization using the history file and the prior narrow date range.

126. The computer program of claim 125 wherein the narrow date range is a concatenation of the current narrow date rang and the prior narrow date range st red in a hist ry fil during th pri r synchronization.

127. The comput r program of claim 125 wherein the instruction for performing a curr nt synchronization further comprises instructions for adding, modifying, or deleting records of the first database that are within the current narrow date range.

128. The computer program of claim 125 wherein the instruction for performing a current synchronization further includes instructions for deleting records of the first database that were present during a previous synchronization and that, following the current synchronization, are outside of the current narrow date range.

129. The computer program of claim 125 wherein the instruction for performing a current synchronization further includes instructions for updating or deleting records of the first and second databases that are outside of the current narrow date range and within the narrow date range, based on the current synchronization.

130. The computer program of claim 125 wherein the instruction for performing a current synchronization further comprises instructions for, based on a selection by a user, performing one of:

(a) deleting the records of the first database that were present during a previous synchronization and that, following the current synchronization, are outside of the current narrow date range, and

(b) updating or deleting records of the first and second databas s that are utsid of the current narrow date range and within the narrow date range, based on the curr nt synchronization.

131. The computer program of claim 125 wherein on of the narrow dat range, the prior narrow date range, and the current narrow date range includes a start date and a stop date and the date of a record being synchronized is compared to the start date and the stop date to determine whether the record is within a corresponding one of the narrow date range, the prior narrow date range, and the current narrow date range.

132. The computer program of claim 131 wherein a record being synchronized includes a record start date and a record stop date, the computer program further comprising instructions for:

performing a comparison of the record start date to the stop date of one of the narrow date range, the prior narrow date range, and the current narrow date range, and of the record stop date to the start date of the corresponding one of the narrow date range, the prior narrow date range, and the current narrow date range;

determining based on the comparison whether the record is within the corresponding one of the narrow date range, the prior narrow date range, and the current narrow date range.

133. The computer program of claim 116 wherein the narrow date range comprises instructions for a relative narrow date range, the relative narrow date range being determined relative to a date of the current synchronization.

134. The computer program of claim 125 wherein the curr nt narr w date range comprises instructions for a r lativ narr w date range, the relativ narrow date range b ing determined relativ t a date of the current synchr nization.

135. The comput r program of claim 116 wherein the first database contains a first plurality of n n-recurring records representing a plurality of recurring date bearing instances, the computer program further comprising instructions for:

generating a synthetic recurring record using the first plurality of non-recurring records;

performing a synchronization of the synthetic recurring record with a record of the second database;

if one of the record, the synthetic record, and a non-recurring record in the first plurality of non-recurring records is outside the narrow date range, fanning the synthetic record, or the record of the second database if it is a recurring record, into a second plurality of non-recurring records within the narrow date range.

136. The computer program of claim 116 wherein the narrow date range comprises instructions for a concatenation of a first date range for the first database and a second date range for the second database.

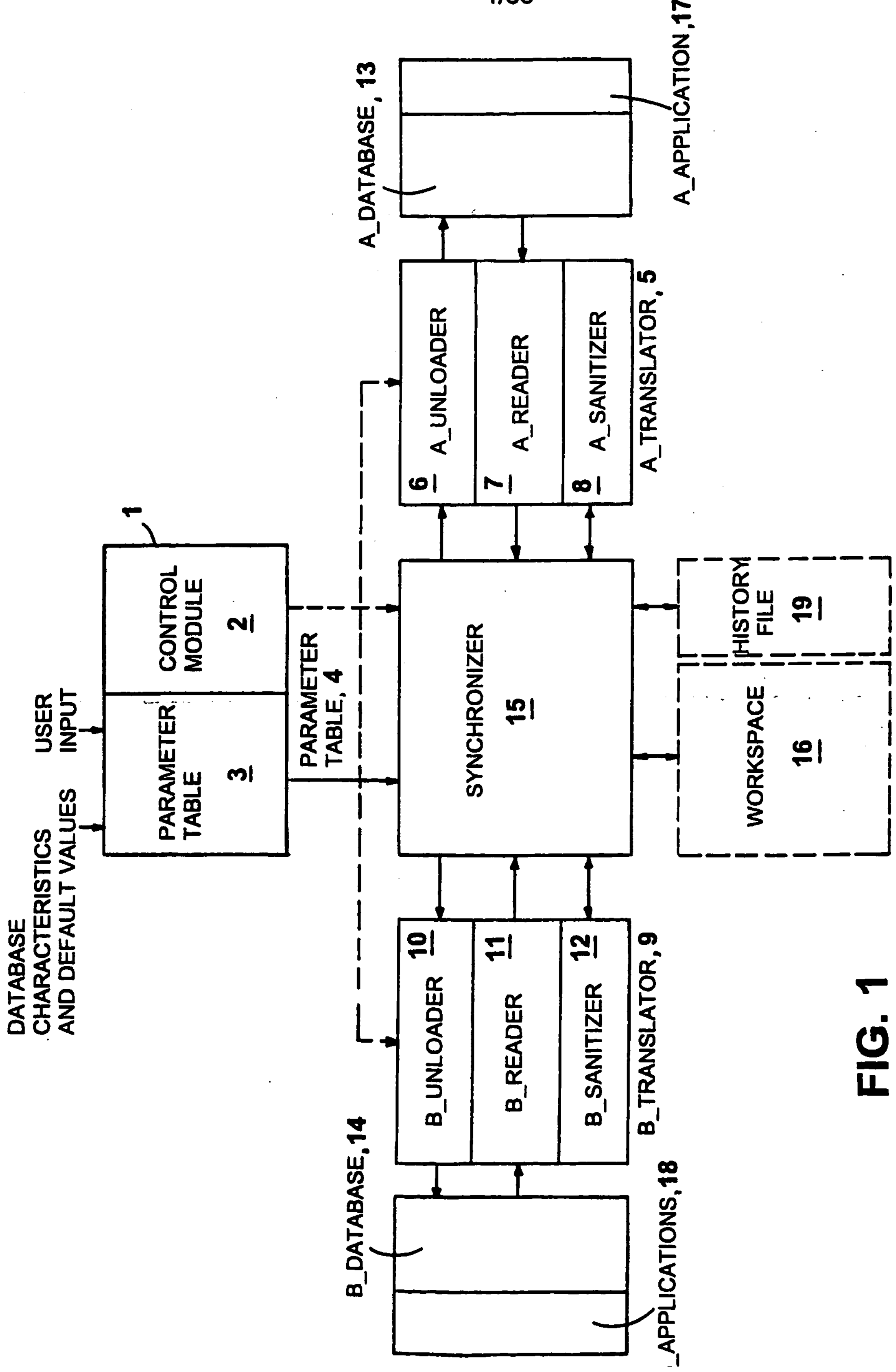

DATABASE CHARACTERISTICS AND DEFAULT VALUES
USER INPUT
1
PARAMETER TABLE 3
CONTROL MODULE 2
PARAMETER TABLE, 4
SYNCHRONIZER 15
A_UNLOADER 6
A_READER 7
A_SANITIZER 8
A_TRANSLATOR, 5
A_DATABASE, 13
A_APPLICATION,17
B_UNLOADER 10
B_READER 11
B_SANITIZER 12
B_TRANSLATOR, 9
B_DATABASE,14
_APPLICATIONS,18
HISTORY FILE 19
WORKSPACE 16

FIG. 1

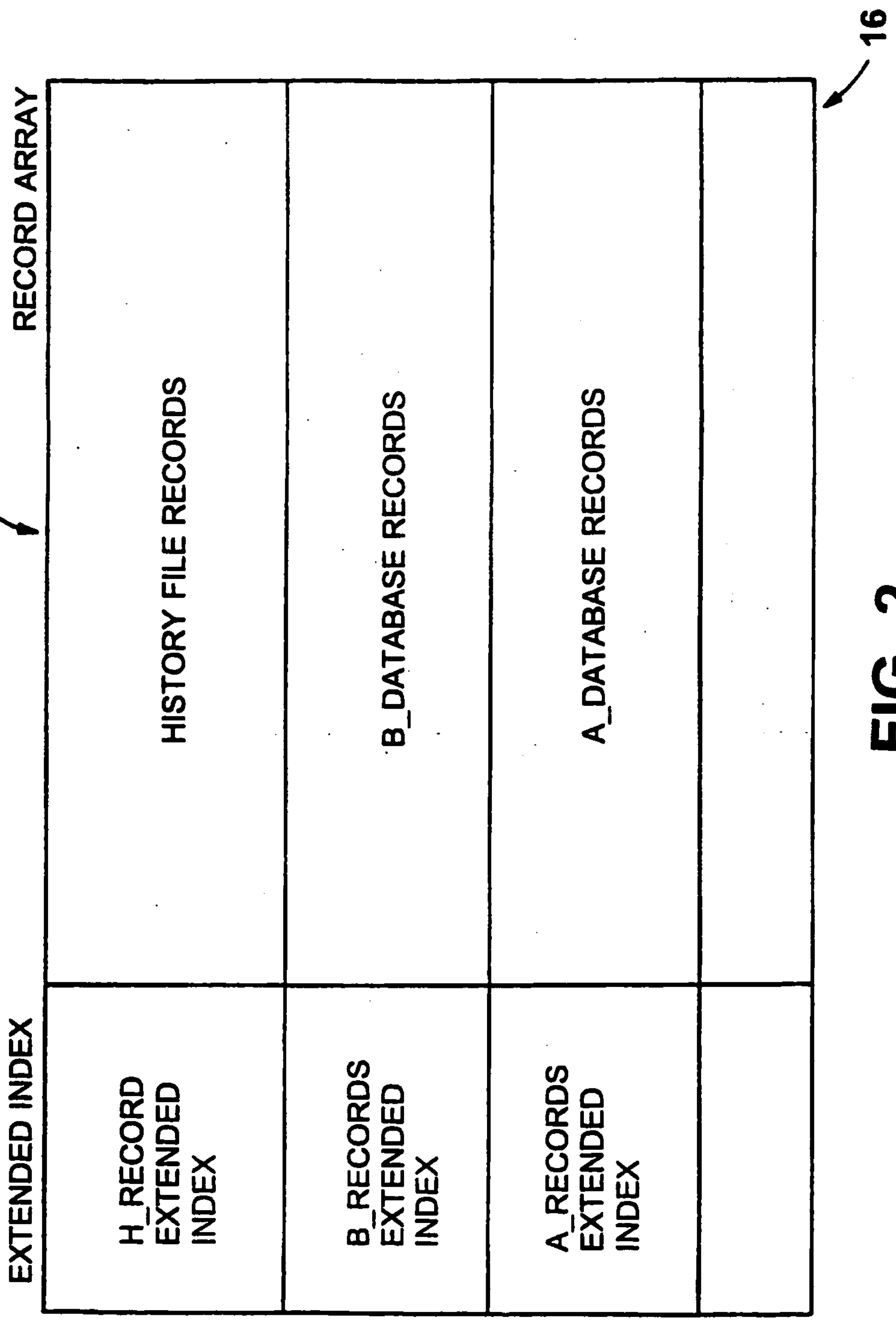

EXTENDED INDEX
RECORD ARRAY
21
16
H_RECORD EXTENDED INDEX
HISTORY FILE RECORDS
B_RECORDS EXTENDED INDEX
B_DATABASE RECORDS
A_RECORDS EXTENDED INDEX
A_DATABASE RECORDS

FIG. 2

Pseudo Code for Translation Engine Control Module

```
100.  CREATE Parameter_Table from User Input A & B database characteristics and default values
101.  INSTRUCT Synchronizer to initialize itself
102.  INSTRUCT Synchronizer to LOAD the History_File into its WORKSPACE
103.  INSTRUCT B_Translator to LOAD all of B_records from B_Database and SEND to Synchronizer
      (Synchronizer STORES these records in WORKSPACE)
104.  INSTRUCT A_Translator to SANITIZE B_records that were just LOADED (A_Translator USES
      Synchronizer services to read and write records in the WORKSPACE; Synchronizer maps these records
      using the B-A_Map before sending them to A_Translator and maps them back using A-B_Map before
      rewritting them into the WORKSPACE)
105.  INSTRUCT A_Translator to LOAD all of A_records form A_Database and SEND to Synchronizer
      (Synchronizes STORES these records in WORKSPACE by first mapping them using the A-B_Map and
      them storing in their new form)
106.  INSTRUCT B_Translator to SANITIZE A_records that were just LOADED (B_Translator uses
      Synchronizer services to read and write records in the WORKSPACE)
107.  INSTRUCT Synchronizer to do CAAR (Conflict Analysis And Resolution) on all the records in
      WORKSPACE.
108.  INFORM user exactly what steps Synchronizer proposes to take (i.e. Adding, Changing, and Deleting
      records). WAIT for User
109.          IF user inputs NO, THEN ABORT
110.  INSTRUCT B_Translator to UNLOAD all applicable records to B_Database.
111.  INSTRUCT A_Translator to UNLOAD all applicable records to the A_Database.
112.  INSTRUCT Synchronizer to CREATE a new History File.
```

FIG. 3

Pseudocode for Generating Parameter Table

```
{Get Input from the user}
150.    ASK user to whether to synchronize based on a previously stored set of prefernces
        (Previous_Preferences) or based on a set of new preferences (New_Preferences)
151.    IF New_Preferences THEN
152.            ASK user whether Incremental_Synchornization or Synchronization_from_Scratch
153.            ASK user following information and STORE in Parameter_Table
        a.      A_Application and B_Application Names
        b.      ADB and BDB Names
        c.      ADB and BDB Locations
        d.      Which sections to Synchronize
        e.      Conflict Resolution Option: IGNORE, ADD, DB WINS, BDB WINS, or NOTIFY
        f.      Other user preferences
154.            ASK user whether wants default mapping for the selected sections of the two databases or wants
                to modify default mapping
155.            LOAD A_Database-B_Database (2)
156.            IF Default_Mapping THEN
157.                    STORE A-B_Map AND B-A_Map in Parameter_Table
158.            END IF
159.            IF Modified_Mapping THEN
160.                    DISPLAY A-B_Map and B-A_Map
161.                    ASK user to modify Maps as desired
162.                    STORE the new A-B_Map and B-A_Map in the Parameter_Table
163.            END IF
165.            END IF
```

FIG. 4A

```
166.    IF Previous_Preferences THEN
167.            ASK user whether Incremental_Synchornization or Synchronization_from_Scratch
168.            STORE in Parameter_Table
169.            LOAD Previous Preferences regarding which databases, mapping, and so on
170.            STORE in the Parameter_Table
171.    END IF
        {User now specifies Date Range}
172.    ASK user to choose Date Range Option
        a.      Previously chosen  Automatic_Date_ Range calculated from today
        b.      Input New Automatic_Date_Range
        c.      Input static Date Range for this Synchronization
        d.      All dates
173.    CALCULATE Start_Current_Date_Range and End_Current_Date_Range based on vlaues from step 171
174.    STORE in Parameter_Table
175.    LOAD parameters setting out characteritics of A_Database and B_Database from Parameters database,
        including
        a.      Field_List_A and Field_List_B
        b.      A_Translator and B_Translator Module Identifiers
        c.      ADB_Section_Names and BDB_Section_Name
176.    STORE in Parameters Table
```

FIG. 4B

```
200.    RECEIVE following from Parameter Table
        1) Name of A_App
        2) Name of B_App
        3) Name and Location  of A_DB
        4) Name and Location of B_DB
        5) Section name of A_Application to be synchronized
        6) Section name of B_Application to be sy
        7) Incremental_Synchronization or Synchronization_From_Scratch Flags
201.    SEARCH for H_File matching Parameters 1-6
202.    If Found H-File and Incremental_Synchrnization THEN DO nothing
203.    IF Found H-File and Synchrnization_from_Scratch, THEN DELETE H_File
204.    IF NOT found H-File, THEN SET Synchronization_from_Scratch AND ASSIGN file name for history
        file.
205.    LOAD from Parameter_Table Start_Current_Date_Range and End_Current_Date_Range
206.    LOAD from Parameter_Table Field_Lists for A-DB and B-DB and field and mapping information
207.    If Incremental_Synchronization THEN COMPARE Field_Lists and Maps from Parameter_Table with
        History_Field_Lists and Maps
208.    IF exact match THEN DO nothing
209.    IF not exact match THEN DELETE H_file AND SET Synchrnization_from_Scratch
210.    CREATE WORKSPACE using Field_List_B
211.    If Incremental_Synchronization THEN Copy H_file into WORKSPACE
212.    FOR each H-Record update
{analyze & update source of extended index}
213.            Do Nothing to NEXT_IN_FIG
214.
```

FIG. 5A

```
215.         FIND H-Record with matching KeyFields
216.         IF FOUND THEN Update NEXT_IN_SKG of H-Record
217.         IF Appointment type and Non-Recurring record THEN
218.             IF (Start_Date after End_Previous_Date_Range) OR (End_Date before
                 Start_Previous_Date_Range) THEN SET Bystander Flag END IF
219.             IF (Start_Date after End_Current_Date_Range) OR (End_Date before Start_Current
                 _Date_Range) THEN SET Outside_Current_Range END IF
{Recurring records}
220.             ELSE
221.             Fan_Out_Recurrence_Pattern for H-Record
222.             SET Bystander Flag and Outside_Current_Range Flags for H-Record
223.             For all Fanned out Instances
224.                 IF (Start_Date Before End_Previous_Date_Range) OR (End_Date after
                     Start_Previous_Date_Range) THEN UN-SET Bystander Flag of Recurring H-
                     Record END IF
225.                 IF (Start_Date before End_Current_Date_Range) OR (End_Date after
                     Start_Current _Date_Range) THEN UN-SET Outside_Current_Range END IF
226.             END LOOP
227.         END IF
228.     END LOOP
```

FIG. 5B

```
235.    LOAD Rep_Basic, Start_Date, Stop_Date, Frequency
236.    CALCULATE Useful Start_Date and Useful_Stop_Date based on Start_Date, Stop_Date, Max_Fan_Out
        and Usefulness_Range_Future & Past
237.    REPEAT
238.            CALCULATE Next_Date based on Useful_Start_Date, Current_Date, Rep_Basic, Frequency,
                Max_Fan_Out
239.            IF Next_Date After Useful_Stop_Date, THEN EXIT
240.            STORE Next_Date
241.                    Fan_Out_Date_Array
242.            Current_Date = Next_Date
243.    END LOOP
```

FIG. 6

Pseudocode for Key_Field_Match

```
250.    RECEIVE Key_FIeld_Hash and WORKSPACE_ID
251.    For all records in WORKSPACE
252.            IF Match_Hash_Value equals Hash Values of Record THEN LOAD the two records
253.                    COMPARE the key fields two records
254.                    IF Exact Match THEN SET Match_Found
255.                    EXIT LOOP
256.            END IF
257.    END LOOP
258.    If Match_Found THEN SEND Success Flag and WORKSPACE ID of Matching record
```

FIG. 7

Pseudo Code for Loading Records of B_database into WORKSPACE

```
B_Translator:

300.    FOR ALL Records in B_DB
301.            READ Record from B_DB
302.            IF (record outside of combination of Current_Date_Range and Prevous_Date_Range), THEN
                GOTO END LOOP
303.            IF NOT right origin tag for this synchronization THEN GOTO END LOOP
304.            SEND Record to Synchronizer 325-236
305.    END LOOP

Synchronizer:
325.    RECEIVE B_Record
326.    STORE in WORKSPACE in next available space
```

FIG. 8

Pseudo Code for Generic A_Sanitization of B_DB Records in Workspace

A_Translator:

```
350.  REPEAT
351.          FOR EVERY Field in an A_Record
352.                  REQUEST Field from Synchronizer
353.                  IF Last_Field, THEN EXIT LOOP
354.                  SANITIZE Field, according to A_Sanitization rules
355.          END LOOP
356.          IF Last_Field, THEN EXIT LOOP
357.          SANITIZE Record according to A_Sanitization rule
358.          FOR EVERY Field in an A_Record
359.                  SEND Field value to Sanitizer
360.          END FOR
361.  UNTIL EXIT
```

SYNCHRONIZER:

```
375.  In Response to Request for Field by A_Sanitizer
376.  REPEAT UNTIL LAST RECORD
377.          READ B_Record
378.          MAP Record according to B_A Map
379.          REPEAT UNTIL A_Translator Request a field from a new Record
380.                  SEND REQUESTED B_field to A_Translator
381.                  WAIT FOR RETURN of B_Field from A_Translator
382.                  STORE field Value in Mapping_Cache
383.          END LOOP
384.          MAP record in Cache according to A→B Map
385.          STORE record in WORKSPACE
386.  END LOOP
387.  SEND Last_Field flag in response to REQUEST
```

FIG. 9

Specific Example of Sanitization

```
400.    IF StartDate and EndDate are both blank
401.            Make Alarm Date blank and make Alarm Flag = FALSE
402.      ELSE IF EndDate is blank THEN SET EndDate = StartDate
403.            ELSE IF StartDate is blank OR is greater than EndDate THEN       SET StartDate =
                EndDate END IF

404.            IF AlarmFlag is TRUE and AlarmDate is blank THEN SET  AlarmDate = StartDate
405.                    ELSE IF AlarmDate is greater than EndDate THEN SET AlarmDate = EndDate
406.    END IF
```

FIG. 10

Pseudocode for Analyzing ID_bearing FIGs

```
550.  FOR EVERY Recurring Master of ID_Bearing FIGS in H_file
551.        FOR EVERY FIG H_Record in Recurring Master FIG
552.              REMOVE Record from SKG it belongs to
553.              IF Record is a singleton CIG, THEN ADD to New_Exclusion_List
554.              IF Record is a doubleton CIG, THEN
555.                    IF the two Records in CIG are Identical, THEN remove other RECORD from
                        its SKG
556.                    END IF
557.                    ELSE IF the two records are NOT Identical, THEN ADD FIG record to
                        New_Exclusion_List and change records into singleton CIGs
558.              END IF
559.        END LOOP
560.        CREATE Synthetic Master record entry in WORKSPACE
561.        COPY value from one of the CIG mates into Synthetic Master
562.        COPY Rep Basic (i.e. recurrence pattern) from the Recurring Master into Synthetic Master
563.        COPY Exclusion List from the database Recurring Master into Synthetic Master and MERGE
            with New_Exclusion_List
564.        COMPUTE all Hash values for Synthetic Master
565.        CREATE new FIG between Synthetic Master the CIGmates of the H-FIG records
566.        CREATE CIG among the three Recurring Masters

{Fan Out Creep}

567.        Fan out Recurring Master with Previoius_Date_Range
568.        Fan out Recurring Master with Current_Date_Range
569.        IF two date arrays are NOT identical, THEN MARK CIG with Fan_Out_Creep flag
570.        MARK all Records in H_File Recurring Master FIG and Synthetic Master FIG as
            Dependent_FIG
571.  END LOOP
```

FIG. 13

Pseudo Code for EXPANDING ID_BASED CIGs

```
600.  For each H_record,
601.          IF single record CIG, THEN GO TO END LOOP
602.          IF triple record CIG, THEN REMOVE CIG records from their SKGs
603.          IF Dependant_FIG, THEN GO TO END LOOP
604.          IF record needed to make triple has to be from a DB with unique ID, THEN GO TO END
              LOOP
605.          For all members of SKG to which H_record belongs
606.                  IF Non_Key_Field_Hash of H_record and SKG_record Match, THEN
607.                          IF Exact Match of all fields with H item THEN Strong_Match is found  END
                              IF
608.                  ELSE
609.                          IF H_Record is a Recurring Master, THEN Find Fanned Instance (Table
                              Recurring Master/Instance Match) which is Strong_Match
610.                  END IF
611.          END LOOP
612.          IF Strong_Match is found AND IF the SKG_Record is Weak_Match member of a CIG, THEN
613.                  REMOVE SKG Record from Weak_Match CIG AND Seek Alternate Weak_Match for
                      the CIG
614.                  ADD SKG record to Current doubleton CIG AND Record for the Weak_Match_CIG
615.                  REMOVE all records in CIG from SKG
616.          END IF
617.          IF Strong Match is NOT found, THEN FIND Weak_Match
618.          IF Weak Match is found, THEN create Weak_CIG
619.                  ELSE REMOVE all records in CIG from SKG
620.          END IF
621.  END LOOP
```

FIG. 14

Pseudo Code for Finding Weak Matches for a Record

```
622.  FOR EVERY Record in SKG
623.        IF (SKG record is from same database as records for which match is sought OR
624.              SKG record already is a Weak_Match record in a CIG OR
625.              SKG record is a Dependent_FIG OR
626.              SKG record is Non_Recurring AND records for which is sought are not, OR
627.              SKG record is Recurring AND records for which is sought are not)
628.        THEN
629.              GO TO END LOOP
630.        ELSE
631.              Ifrecurring item OR Key_Date_Field match Exactly, THEN Weak_Match is found
632.        END IF
633.  END LOOP
```

FIG. 15

Pseudo Code for Finding Matches between Recurring items and Non_Unique ID Bearing Instances

```
650.  IF Instances' database does not have unique ID OR synchronizing from scratch THEN CONTINUE
651.          ELSE EXIT
652.  END IF
653.  FOR any Recurring_Master not in Instances database,
654.          Fan out Recurring_Master for Previous_Date_Range into Previous_Date_Array
655.          MARK all entry as Previous_Date_Range_Instance
656.          Fan out Current_Recurring_Master for Current Data Range into Current_Dates_Array
657.          MARK all entries as Current_Date_Range_Instance
658.          MARK records in Exclusion_List as EXCLUDED_Dates
659.          MERGE Exclusion_List, Previous_Date_Array and Current_Date_Array into
              Merged_Date_Array
660.          CREATE Slave_Date_Array
661.          FOR EVERY item in SKG of Recurring_Master
662.                  IF Recurring item OR NOT Instances database record, THEN GO TO END LOOP
663.                  IF Start_Date of SKG record Matches an Entry in Merged_Date_Array THEN STORE
                      in Slave_Array WORKSPACE record number of SKG record AND
                      Merged_Date_Array in Slave Array
664.          END LOOP
665.          FOR EVERY Unique Non_Date Hash of Slave_Array records
666.                  FIND Slave_Array records with matching Non_Date Hash
667.                  COUNT number of matches
668.          END LOOP
669.          FIND the largest number of match counts
670.          IF largest is less than 30% of number of unexcluded instances of Master Recurring, THEN
              EXIT
```

FIG. 16A

```
671.        IF Match equals one, THEN IF NOT exact match, THEN EXIT
672.        CREATE Homogenous_Instance_Group from the records which have the same Non_Date_Hash
            value as the largest match
673.        CREATE new record Synthetic_Master in WORKSPACE
674.        COPY Basic Repeat Pattern of Recurring_Master into Synthetic Master
675.        COPY Other values from 1st item of Homogeneous Instance Group into Synthetic Master
676.        CREATE Synthetic_Master Exclusion_List based on differences between Merged_Date_Array
            and Homogeneous_Instance_Group
677.        COMPUTE Hash values for Synthetic_Master
678.        ADD Synthetic_Master to CIG of Recurring_Master
679.        CREATE Synthetic_Master FIG from all Homogeneous_Instances_Group item
680.        FOR EVERY Homogeneous_Instances_Group_item,
681.            IF Weak_match in another CIG, THEN REMOVE from CIG AND FIND New WEAK
                MATCH for that CIG
682.            REMOVE from its SKG
683.            MARK as Dependant_FIG
684.        END LOOP
685.        IF dates in Previous_Date_Array which are not in Current_Date_Array OR Vice_versa THEN
            MARK CIG Fan_Out_Creep Flag (for unload time)
686.    END LOOP
```

FIG. 16B

Pseudocode for Completing SKG Analysis

```
700.  IF A_database AND B_database are unique ID bearing DBs, THEN REMOVE ALL remaining H_items
      from SKGs
702.  END IF
703.  FOR ALL SKGs in WORKSPACE
704.        IF SKG is singleton, THEN GO TO END LOOP
705.        FOR ALL items in Current_SKG
706.              IF item is Weak_Match AND part of ID_based pair, THEN REMOVE from SKG
707.        END LOOP
708.        FOR ALL records in Current_SKG begining with H_Records
709.              Call Set CIG_Max_Size in F igure 18
710.              FIND Strong Match or Master/Instance Match between Non_ID bearing database
                  record and H_Records
711.              IF FOUND, THEN ADD to CIG
712.                    ELSE IF FIND Strong_Match in SKG between BA and B database records
                        THEN Attach records together as CIG END IF
713.              END IF
714.              IF CIG_Size = CIG_MAX_Size, THEN REMOVE ALL CIG members from SKG
715.        END LOOP
716.        IF CIG_Max_Size = 3, THEN
717.              FOR EVERY two record CIG in SKG,
718.                    FIND Weak_Match (Same Key_Date_Field and Same Recurrence Level)
719.                    IF Weak_Match item from opposing DB, THEN ADD to CIG
720.                    REMOVE records in CIG from SKG
721.              END LOOP
722.        END IF
723.        FOR EVERY SKG item
724.              FIND Weak_Match (Same Key_Date_Field and Same Recurrence Level)
725.              IF FOUND, THEN ADD to CIG and REMOVE from SKG
726.        END LOOP
727.  END LOOP
```

FIG. 17

Pseudocode for setting Maximum CIG Size for Every CIG analyzed in Fig. 17.

```
750.    CIG_Max_Size = the number of non-unique ID bearing applications +1
751.    If the CIG_Max_size = 1 and CIG is not a H_Record THEN CIG_MAX_Size = 2
```

FIG. 18

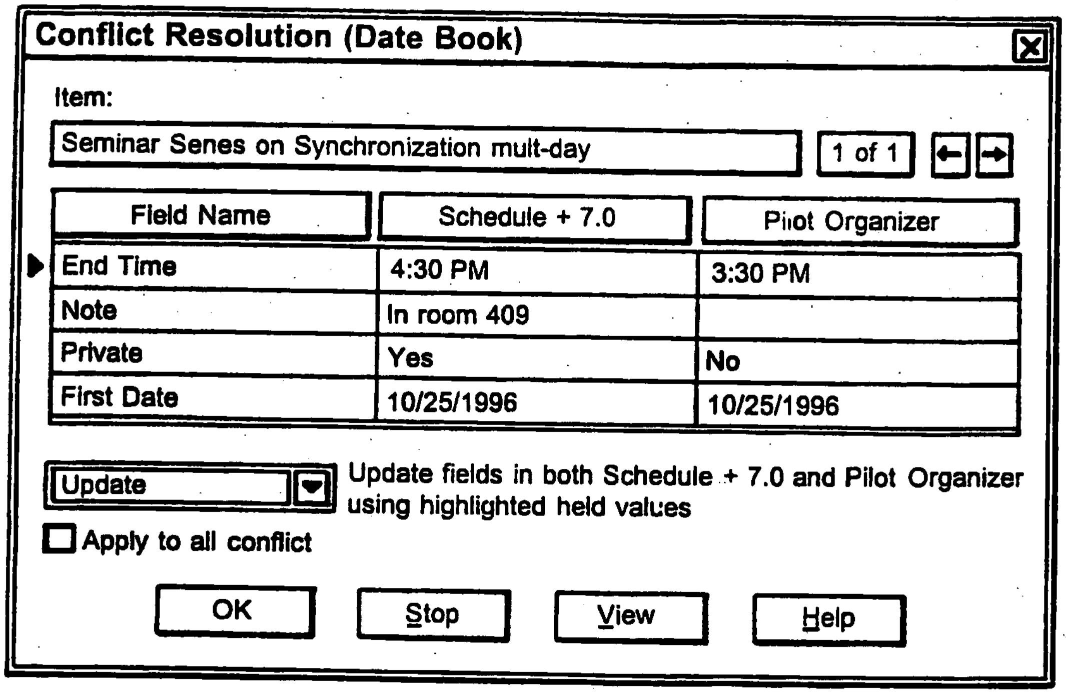
Pseudo Code for setting CIG types

```
800.  FOR EVERY CIG
801.      IF CIG Size is 1, THEN
802.          DETERMINE origin of the CIG record
803.          IF H_Record, THEN CIG_Type = 010
804.          IF B_Record, THEN CIG_Type = 001
805.          IF A_Record, THEN CIG_Type = 100
806.      END IF
807.      IF CIG Size is 2, THEN
808.          COMPARE the two CIG records
809.          IF two members are the same, THEN
810.              DETERMINE the origin of the CIG records
811.              IF B_Record and H_Record, THEN CIG_Type = 011
812.              IF A_Record and H_Record, THEN CIG_type = 110
813.              IF B_Record and A_Record, THEN CIG_type = 101
814.          END IF
815.          IF two records are different, THEN
816.              DETERMINE the origin of the CIG records
817.              IF B_Record and H_Record, THEN CIG_Type = 012
818.              IF A_Record and H_Record, THEN CIG_type = 210
819.              IF B_Record and A_Record, THEN CIG_type = 102
820.          END IF
```

FIG. 19A

```
821.         END IF
822.         IF CIG_Size = 3, THEN
823.               COMPARE records
824.               DETERMINE origins of records
825.               IF ALL records are the same, THEN CIG_Type = 111
826.               IF A_Record different from the other two and B_Record= H_Record,      THEN
                   CIG_Type = 211
827.               IF B_Record different from the other two and A_Record= H_Record,      THEN
                   CIG_Type = 112
828.               IF H_Record different from the other two and B_Record= A_Record,      THEN
                   CIG_Type = 212
829.               IF ALL records are different, THEN CIG_Type = 213
830.         END IF
831.   END LOOP
```

FIG. 19B



Conflict Resolution (Date Book)

Item:

Seminar Senes on Synchronization mult-day

1 of 1

Field Name	Schedule + 7.0	Pilot Organizer
End Time	4:30 PM	3:30 PM
Note	In room 409	
Private	Yes	No
First Date	10/25/1996	10/25/1996

Update

Update fields in both Schedule + 7.0 and Pilot Organizer using highlighted held values

☐ Apply to all conflict

FIG. 20

Pseudocode for Merging Exclusion Lists

```
850.    FOR ALL Recurring Masters,
851.            IF CIG_Type is 102 and conflict is unresolved THEN GO TO END LOOP
{Changing CIG TYPE}
852.            COMPARE Exclusion_Lists of Current_CIG A and B records to determine Exclusion instances
                which appear in only one of the two records (i.e. One_Side_Only_Exclusion)
853.            IF None THEN do nothing
854.                    ELSE IF One_side_only_Exclusion in A_Record but not in B THEN USE Table in
                        FIG. 22 to Convert CIG_Type
855.                    ELSE IF One_Side_Only_Exclusion in B record but not in A THEN USE Table in
                        FIG. 23 to Convert CIG_Type
856.                    ELSE IF One_Side_Only_Exclusion in both records, THEN USE Table in FIG. 24 to
                        convert CIG_Type
857.            END IF
858.    END LOOP
```

FIG. 21

Old CIG + choice	new CIG	new Conflict Resolution Choice	Other Instructions & Comments
101	102	ADB Wins	
111	211		
112	132		Replace H_Record with a copy of the B_Record, plus the ADB Exclusion List
211	211		
212	213	ADB Wins	
132	132		Copy ADB ExclusionList into P-Item
102-Ig	102	Ignore	
102-SW	102	ADB Wins	
102-TW	132		Create H_Record by copying the B_Record, plus the ADB Exclusion List
213-Ig	213	ADB Wins, Excl Only	The Excl Only flag is set so that only the Exclusion List will be updated. Other BDB Fields will remain unchanged.
213-SW	213	ADB Wins	
213-TW	132		Replace P-Item with a copy of the B_Record, plus the ADB Exclusion List

(Ig for Ignore, SW for ADB Wins, or TW for BDB Wins).

FIG. 22

Old CIG + choice	new CIG	new Conflict Resolution Choice	Other Instructions & Comments
101	102	BDB Wins	
111	112		
112	112		
211	132		Replace P-Item with a copy of the A_Record, plus the BDB Exclusion List
212	213	BDB Wins	
132	132		Copy BDB ExclusionList into P-Item
102-Ig	102	Ignore	
102-SW	132		Create P-Item by copying A_Record, plus the BDB Exclusion List
102-TW	102	BDB Wins	
213-Ig	213	BDB Wins, Excl Only	The Excl Only flag is set so that only the Exclusion List will be updated. Other ADB Fields will remain unchanged.
213-SW	132		Replace P-Item with a copy of the A_Record, plus the BDB Exclusion List
213-TW	213	BDB Wins	

(Ig for Ignore, SW for ADB Wins, or TW for BDB Wins)

FIG. 23

Old CIG + choice	new CIG	new Conflict Resolution Choice	Other Instructions & Comments
101	132		Create P-Item by copying B_Record, plus the Merged Exclusion List
111	132		Copy Merged Exclusion List into P-Item.
112	132		Replace P-Item with a copy of the B_Record, plus the Merged Exclusion List
211	132		Replace P-Item with a copy of the A_Record, plus the Merged Exclusion List
212	132		Replace P-Item with a copy of the B_Record, plus the Merged Exclusion List
132	132		Copy Merged ExclusionList into P-Item
102-Ig	102	Ignore	
102-SW	132		Create P-Item by copying A_Record, plus the Merged Exclusion List
102-TW	132		Create P-Item by copying B_Record, plus the Merged Exclusion List
213-Ig	132	Excl Only	Copy Merged ExclusionList into P-Item. The Excl Only flag is set so that only the Exclusion List will be updated. Other ADB and BDB Fields will remain unchanged.
213-SW	132		Replace P-Item with a copy of the A_Record, plus the Merged Exclusion List
213-TW	132		Replace P-Item with a copy of the B_Record, plus the Merged Exclusion List

(Ig for Ignore, SW for ADB Wins, or TW for BDB Wins)

FIG. 24

Pseudo Code for Unloading Records from WORKSPACE to a database for non_rebuild_all database

```
899.  FOR all Recurring Masters which require Fanning and Outcome is UPDATE or DELETE, call
        Synchronizer Function Fanning for Unloading, Fig.27
900.  COUNT RECORDS to be Unloaded by examining all CIGs
901.    FOR EVERY RECORD to be Unloaded
{DETERMINE OUTCOME}
902.      IF MARKED GARBAGE, THEN SKIP
903.      IF BYSTANDER AND NOT History File Unload, THEN SKIP
904.      IF WRONG_SUBTYPE AND NOT Rebuild_All Translator, THEN SKIP
905.      IF Recurring_Master THEN IF Fanned for the database THEN UNLOAD Instances when
          unloading END IF
906.        ELSE UNLOAD Recurring Master when unloading
907.      END IF
908.      LOOK UP Outcome_Sync (i.e., Unload Instructions) in Fig. 26 Table based on CIG_TYPE]
909.      IF Date Range Limited Database and Date_Range_Option = LENIENT, THEN
910.        IF RECORD is Out of Current_Date_Range AND Outcome is not DELETE, THEN
            SKIP Record
911.      ELSE IF Date Range Limited Database and Date_Range_Option = STERN, THEN
912.        IF RECORD is Out of Current_Date_Range, THEN Outcome=DELETE
913.      END IF
914.      IF Outcome = DELETE, THEN
915.        Get Info Required for this database to DELETE RECORD
916.        (may include unique ID, Record ID, or the original values of one or more key fields, to
            look up record so that it can be deleted)
917.        DELETE Record
918.        SEND Synchronizer SUCCESS/FAILURE FLAG
919.      END IF
```

FIG. 25A

```
920.        IF Outcome = ADD, THEN
921.                GET Current values of all Fields, from Synchronizer
                    (Synchronizer maps for A database based on B-A, in response to each request)
922.                CREATE new RECORD in DB
923.                IF Unique_ID DB, THEN GET Unique_ID
924.                SEND to Synchronizer (Success FLAG with any Unique_ID) OR (Failure Flag)
925.                Synchronizer: Store Unique_ID in WORKSPACE
926.        END IF
927.        IF Outcome is UPDDATE THEN GET Current values to be unloaded and original values loaded
            from database from Synchronizer
928.                COMPARE and DETERMINE which Field to be updated
929.                UPDATE fields in the record to be updated
930.                SEND to Synchronizer (Success flag AND Unique_ID) OR (Failure Flag)
931.                Synchronizer:  STORE Unique_ID in WORKSPACE
932.        END IF
933.    END LOOP
```

FIG. 25B

```
// Original  Current
//  Item      Item     Outcome
// -------   -------   -------
{

//--- TIFCIG_001 - 1  (0) // item is present in BDB only

    B,           B,          oLEAVE_ALONE,    // unloading to BDB
    B,           B,          oADD,            // unloading to ADB
   B,       B,     oSAVE,        // unloading to History File

//--- CIG_100 - 1  (1) // item is present in ADB only

   A_       A_     oADD,          // unloading to BDB
   A_       A_     oLEAVE_ALONE,  // unloading to ADB
   A_       A_     oSAVE,         // unloading to History File

//--- CIG_101 - 1  (2) // item is identical in ADB and BDB

   B_       B_     oLEAVE_ALONE,  // unloading to BDB
   A_       A_     oLEAVE_ALONE,  // unloading to ADB
   A_       B_     oSAVE,         // unloading to History File

//--- CIG_102 - 1  (3) // NEW ADB ITEM < > NEW BDB ITEM
                       // (the BDB WINS outcome is shown here)

   B_       B_     oLEAVE_ALONE,  // unloading to BDB
   A_       B_     oUPDATE,       // unloading to ADB
   A_       B_     oSAVE,         // unloading to History File

//--- CIG_111 - 1  (4) // item is unchanged across the board

   B_       B_     oLEAVE_ALONE,  // unloading to BDB
   A_       A_     oLEAVE_ALONE,  // unloading to ADB
   H_       H_     oSAVE,         // unloading to History File

//--- CIG_112 - 1  (5) // item CHANGED in BDB since last sync

   B_       B_     oLEAVE_ALONE,  // unloading to BDB
   A_       B_     oUPDATE,       // unloading to ADB
   H_       B_     oSAVE,         // unloading to History File

//--- CIG_110 - 1  (6) // item DELETED from BDB since last sync

   H_       H_     oLEAVE_DELETED, // unloading to BDB
   A_       A_     oDELETE,        // unloading to ADB
   H_       H_     oDISCARD,       // unloading to History File

//--- CIG_211 - 1  (7) // item CHANGED in ADB since last sync
```

```
  A_       A_      oLEAVE_ALONE,  // unloading to ADB
  H_       A_      oSAVE,         // unloading to History File

//--- CIG_212 - 1  (8) // item CHANGED IDENTICALLY in Src & BDB

  B_       B_      oLEAVE_ALONE,  // unloading to BDB
  A_       A_      oLEAVE_ALONE,  // unloading to ADB
  H_       A_      oSAVE,         // unloading to History File

//--- CIG_213 - 1  (9) // item CHANGED DIFFERENTLY in Src & BDB
                   // (the BDB WINS outcome is shown here)

  B_       B_      oLEAVE_ALONE,  // unloading to BDB
  A_       B_      oUPDATE,       // unloading to ADB
  H_       B_      oSAVE,         // unloading to History File

//--- CIG_210 - 1  (10) // CHANGED in ADB, DELETED from BDB

  A_       A_      oADD,          // unloading to BDB
  A_       A_      oLEAVE_ALONE,  // unloading to ADB
  H_       A_      oSAVE,         // unloading to History File

//--- CIG_011 - 1  (11) // item DELETED from ADB since last sync

  B_       B_      oDELETE,       // unloading to BDB
  H_       H_      oLEAVE_DELETED, // unloading to ADB
  H_       H_      oDISCARD,      // unloading to History File

//--- CIG_012 - 1  (12) // DELETED from ADB, CHANGED in BDB

  B_       B_      oLEAVE_ALONE,  // unloading to BDB
  B_       B_      oADD,          // unloading to ADB
  H_       B_      oSAVE,         // unloading to History File

//--- CIG_010 - 1  (13) // item DELETED from both ADB & BDB

  H_       H_      oLEAVE_DELETED, // unloading to BDB
  H_       H_      oLEAVE_DELETED, // unloading to ADB
  H_       H_      oDISCARD,      // unloading to History File

//--- CIG_132 - 1  (14) // 102 conflict resolved interactively
                   // to a "compromise" value stored in P-item
                   // outcome is always UPDATE BOTH

  B_       H_      oUPDATE,       // unloading to BDB
  A_       H_      oUPDATE,       // unloading to ADB
  A_       H_      oSAVE,         // unloading to History File

//--- CIG_13F - 1  (15) // 132 UPDATE-BOTH
                   // which has been Fanned To BDB

  B_       B_      oDELETE,       // unloading to BDB
  A_       H_      oUPDATE,       // unloading to ADB
  A_       H_      oSAVE          // unloading to History File
  // Note that we delete the recurring master on the BDB Side;
  // fanned instances take its place.

};
```

The table entries above for CIG_102 and CIG_213 are only relevant when the Conflict Resolution Option is set to BDB WINS. If the Conflict Resolution Option is set to IGNORE or ADB WINS then those table entries are adjusted accordingly. For IGNORE we use the following table entries:

```
// Original  Current
//  Item      Item     Outcome
// -------   -------   -------
//--- _CIG_TYPE_102   // NEW ADB ITEM < > NEW BDB ITEM

    B_        B_       oLEAVE_ALONE,  // unloading to BDB
    A_        A_       oLEAVE_ALONE,  // unloading to ADB
    B_        B_       oDISCARD,      // unloading to History File

//--- _CIG_TYPE_213   // item CHANGED DIFFERENTLY in Src & BDB

    B_        B_       oLEAVE_ALONE,  // unloading to BDB
    A_        A_       oLEAVE_ALONE,  // unloading to ADB
    H_        H_       oSAVE,         // unloading to History File
```

And for ADB WINS we use the following table entries:

```
// Original  Current
//  Item      Item     Outcome
// -------   -------   -------

//--- _CIG_TYPE_102   // NEW ADB ITEM < > NEW BDB ITEM

    B_        A_       oUPDATE,       // unloading to BDB
    A_        A_       oLEAVE_ALONE,  // unloading to ADB
    B_        A_       oSAVE,         // unloading to History File

//--- _CIG_TYPE_213   // item CHANGED DIFFERENTLY in Src & BDB

    B_        A_       oUPDATE,       // unloading to BDB
    A_        A_       oLEAVE_ALONE,  // unloading to ADB
    H_        A_       oSAVE,         // unloading to History File
```

When the NOY option is in effect, CIG-specific conflict outcomes are recorded in the CIG members' flag bits. When this is the case the following lookup table is used:

```
static unsigned char TableAfterILCR [_SYNC_OUTCOME_COUNT]
                                    [AFTER_ILCR_CIG_TYPE_COUNT]
                                    [SYNC_UNLOAD_PHASE_COUNT]
                                    [3] =
// Original  Current
//  Item      Item     Outcome
// ------    ------    ------
{
```

FIG. 26C

```
//------------------ Entries for _OUTCOME_SYNC_BDB_WINS

  //--- _CIG_TYPE_102   // NEW ADB ITEM < > NEW BDB ITEM

    B_      B_      oLEAVE_ALONE,  // unloading to BDB
    A_      B_      oUPDATE,       // unloading to ADB
    A_      B_      oSAVE,         // unloading to History File

  //--- _CIG_TYPE_213   // item CHANGED DIFFERENTLY in Src & BDB

    B_      B_      oLEAVE_ALONE,  // unloading to BDB
    A_      B_      oUPDATE,       // unloading to ADB
    H_      B_      oSAVE,         // unloading to History File

//------------------ Entries for _OUTCOME_SYNC_ADB_WINS

  //--- _CIG_TYPE_102   // NEW ADB ITEM < > NEW BDB ITEM

    B_      A_      oUPDATE,       // unloading to BDB
    A_      A_      oLEAVE_ALONE,  // unloading to ADB
    B_      A_      oSAVE,         // unloading to History File

  //--- _CIG_TYPE_213   // item CHANGED DIFFERENTLY in Src & BDB

    B_      A_      oUPDATE,       // unloading to BDB
    A_      A_      oLEAVE_ALONE,  // unloading to ADB
    H_      A_      oSAVE,         // unloading to History File

//------------------ Entries for IGNORE (LEAVE UNRESOLVED)

  //--- _CIG_TYPE_102   // NEW ADB ITEM < > NEW BDB ITEM

    B_      B_      oLEAVE_ALONE,  // unloading to BDB
    A_      A_      oLEAVE_ALONE,  // unloading to ADB
    B_      B_      oDISCARD,      // unloading to History File

  //--- _CIG_TYPE_213   // item CHANGED DIFFERENTLY in Src & BDB

    B_      B_      oLEAVE_ALONE,  // unloading to BDB
    A_      A_      oLEAVE_ALONE,  // unloading to ADB
    H_      H_      oSAVE          // unloading to History File

}; //---- TableAfterILCR
```

FIG. 26D

```
FANNING Recurring_Items for Unloading (for A DB)

        Fan Pattern for paper Date Range (Fig. XX)

950.    IF Outcome is UPDATE, THEN
951.            IF (CIG A_Record was a Recurring Master but now to be fanned and CIG B_Record is a
                Recurring Master) THEN IF CIG_Type = 132 THEN CIG_Type = 13F
952.                    GOTO Fanning For ADD
953.                    ELSE
954.                            SET A_Record CIG_Type to 100
955.                            SET B_Record CIG_Type to 001
956.                            SET H_Record CIG_Type to 010
957.                            MARK A_Record with DELETE_ME Flag
958.                            GOTO Fanning for Add
959.                    END IF
960.            END IF
961.            IF (CIG A_Records were fanned previously and Fanned now) AND (CIG B_record recurring),
                THEN
962.                    FOR ALL A items in Synthetic Master FIG
963.                            STORE Start_Date in Date_Array_Temporary
964.                    END LOOP
965.                    Fan_Out_Recurring_Pattern of B Master
966.                    COMPARE Date_Array_Temp with Fan_Out_Date_Array
967.                    MARK Dates which NOT IN Fan_Out_Date Array with DELETE_Me Flag
968.                    IF Date NOT IN Date_Array_Temp, THEN
969.                            CREATE WORK SPACE Record by Copy Recurring_Master but Omit Rep
                                Basic, Rep Excl, Unique ID Field
970.                            SET Start_Date, End_Date, Alarm_Date to values for Current Instance
971.                            Compute Hash
972.                            MARK Fanned_For_A
973.                    END IF
```

FIG. 27A

```
974.            IF Date in Date_Array_Temp AND Fan_Out_Date_Array THEN
975.                Compare Non_Date Hash to Synthetic Master Non_Date_Hash
976.                IF Same, THEN MARK Leave_Alone
977.                ELSE MARK UPDATE END IF
978.            END IF
979.        END IF
980.        IF (A_Record Recurring previously and to be Fanned now) AND (CIG B_Record is Instances)
            THEN
981.            MARK CIG items as Garbage
982.            MARK FIG items of CIG H_record as Garbage
983.            MAKE FIG items of CIG B_record singletons
984.        END IF
985.    ELSE [Fanning For Add]
986.        Fan out Recurrence Pattern
987.        For each Date in Fan_Out_Date_Array
988.            COPY Master item into new WORKSPACE Record except Omit Rep_Basic,
                Rep_Exclusion, and Unique ID
989.            Use Date for Start Date and End Date
990.            Set Alarm Date, if necessary
991.            Compute Hash Values
992.            Attach to Recurring_Master FIG
993.            Set Fanned_for_A Flag
994.        END LOOP
995.    END IF
```

FIG. 27B

Pseudocode for Unloading History FILE

```
1000.  ERASE previous History File and CREATE new one
1001.  FOR EVERY CIG in WORKSPACE
1002.          Look up in Fig. 26 Table based on CIG_Type AND DETERMINE whether should be unloaded
               into the History File
1003.          IF NO THEN GOTO END LOOP
1004.          IF Exclusion_List_Only Flag is set when merging of Exclusion_List THEN REPLACE History
               RECORD Exclusion_List with new Merged Exclusion_List
1005.          Clear all Flag bits except for Recurring_Record flag
1006.          SET origin flag to History_Record
1007.          Clear FIG, SKG and CIG words
1008.          STORE Applicable Unique IDs
1009.          IF Recurring item, THEN STORE ALL ID_Bearing FIG records AND SET their FIG in
               History_File to keep them together
1010.          STORE Record in History File
1011.          IF current record is a recurring master for an ID-bearing FIG THEN STORE FIG Records(i.e.
               all Fanned Instances) in the History File, with the FIG linkage words set in the History File to
               hold the FIG together.
1012.  END LOOP
1013.  STORE Field Lists, Application Names, Database Names, Current Date Range,
```

FIG. 28

	How Item is stored in Other Database	How stored in Unloader's Database Before Fanning For Update	How stored in Unloader's Database After Fanning For Update
1	Master	Master	Instances
2	Master	Instances	Instances
3	Instances	Master	Instances

FIG. 29